PARALLEL AND DISTRIBUTED COMPUTING

Project Presentation

L1S23BSCS0257 Faheem Raza

L1S22BSCS0211 Ali Ahmad

L1S22BSCS0188 Abdullah Manzoor

L1S22BSCS0217 Eman Awan

L1S22BSCS0215 Hamza Amir

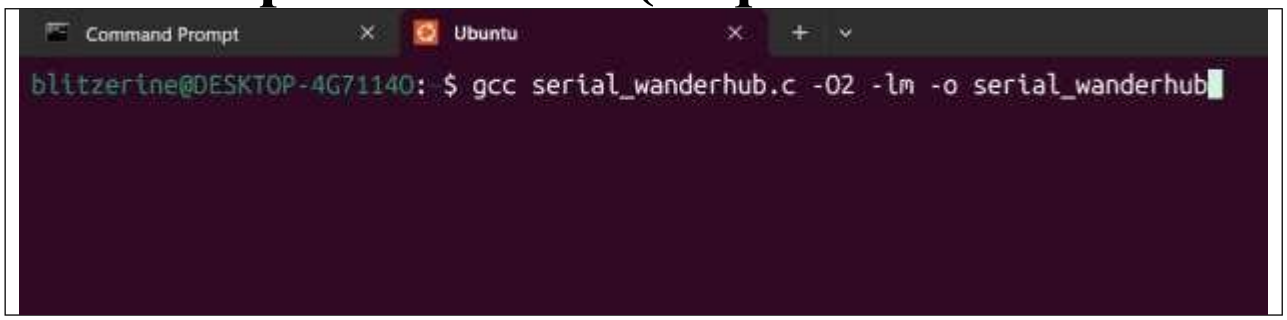
L1S22BSCS0224 Rana Hassan

Parallel and Distributed Computing Project

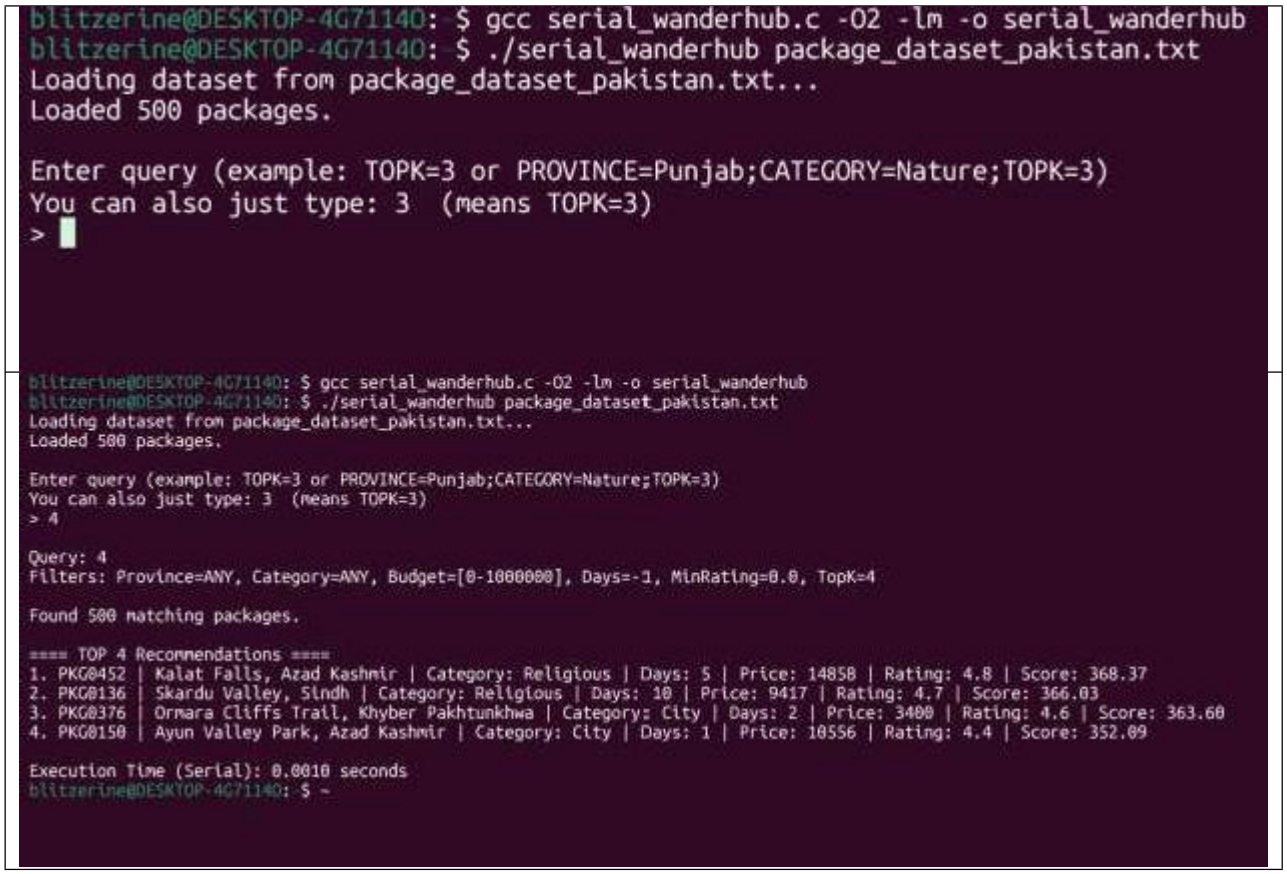
Project Background (Dataset-Based)

This project, *Wander-Hub-AI*, is developed as a **smart travel recommendation system** to demonstrate the practical application of **Parallel and Distributed Computing concepts** using a real-world dataset. The system uses a comprehensive Pakistan-based travel dataset stored in a text file (package_dataset_pakistan.txt), where each record represents a travel package containing attributes such as **package ID, destination name, province, category, duration (days), pricing details, user rating, number of reviews, and popularity score**. As the dataset size grows, sequential processing becomes inefficient; therefore, the project implements multiple execution models including **Serial, Pthreads, OpenMP, MPI, and Client–Server architecture** to efficiently filter, rank, and recommend the best travel packages based on user preferences. By processing the same dataset across different computational models, the project highlights how parallelism and distributed execution significantly reduce execution time while maintaining result accuracy, making it suitable for both academic demonstration and scalable real-world recommendation systems.

Serial Implementation (Sequential Execution Method)



-O2 → Optimization Level 2
-lm → Link the Math Library



1. parse_line()

What it does:
Reads **one row** from the dataset file and saves its data.
In simple words:

- It takes one line from the file
- Splits it using **tab (\t)**
- Stores values like package ID, place, province, price, rating, etc.
- Skips the **header row**

Used for: Loading dataset into memory.

---

2. reset_query_defaults()

What it does:
Resets all search filters to default values.
In simple words:

- Clears province and category
- Sets large budget range
- Sets TOPK = 5
- Removes day and rating filters

Used for: Making sure old queries do not affect new ones.

---

3. is_only_number()

What it does:
Checks if user input contains **only numbers**.
In simple words:

- If user types 3, it returns true
- If user types TOPK=3, it returns false

Used for: Treating input like 3 as **TOPK = 3**.

---

4. parse_query()

What it does:
Reads the user's query and sets filters.
In simple words:

- First resets all filters
- If user types only a number → sets TOPK
- Otherwise reads filters like:
  - Province
  - Category
  - Budget
  - Days
  - Rating
  - TopK

Used for: Understanding what the user wants to search.

---

5. matches_filter()

What it does:
Checks if a package matches the user query.
In simple words:

- Checks province match
- Checks category match
- Checks budget range
- Checks number of days
- Checks minimum rating

Returns:

- YES (1) → package is valid
- NO (0) → package is rejected

---

6. calculate_score()

What it does:

Gives a **score** to each package.

In simple words:

- Higher rating → higher score
- More popularity → higher score
- More reviews → higher score
- Price close to budget → extra score
- Days close to required → extra score

Used for: Ranking packages.

---

7. sort_topk()

What it does:

Sorts packages by score (highest first).

In simple words:

- Uses bubble sort
- Best package comes at the top
- Worst package goes down

Used for: Showing best recommendations first.

---

8. main()

What it does:

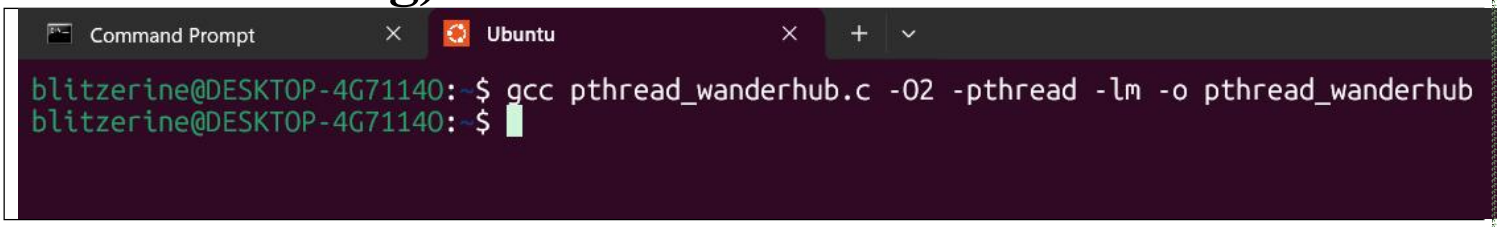
Controls the **whole program**.

In simple words:

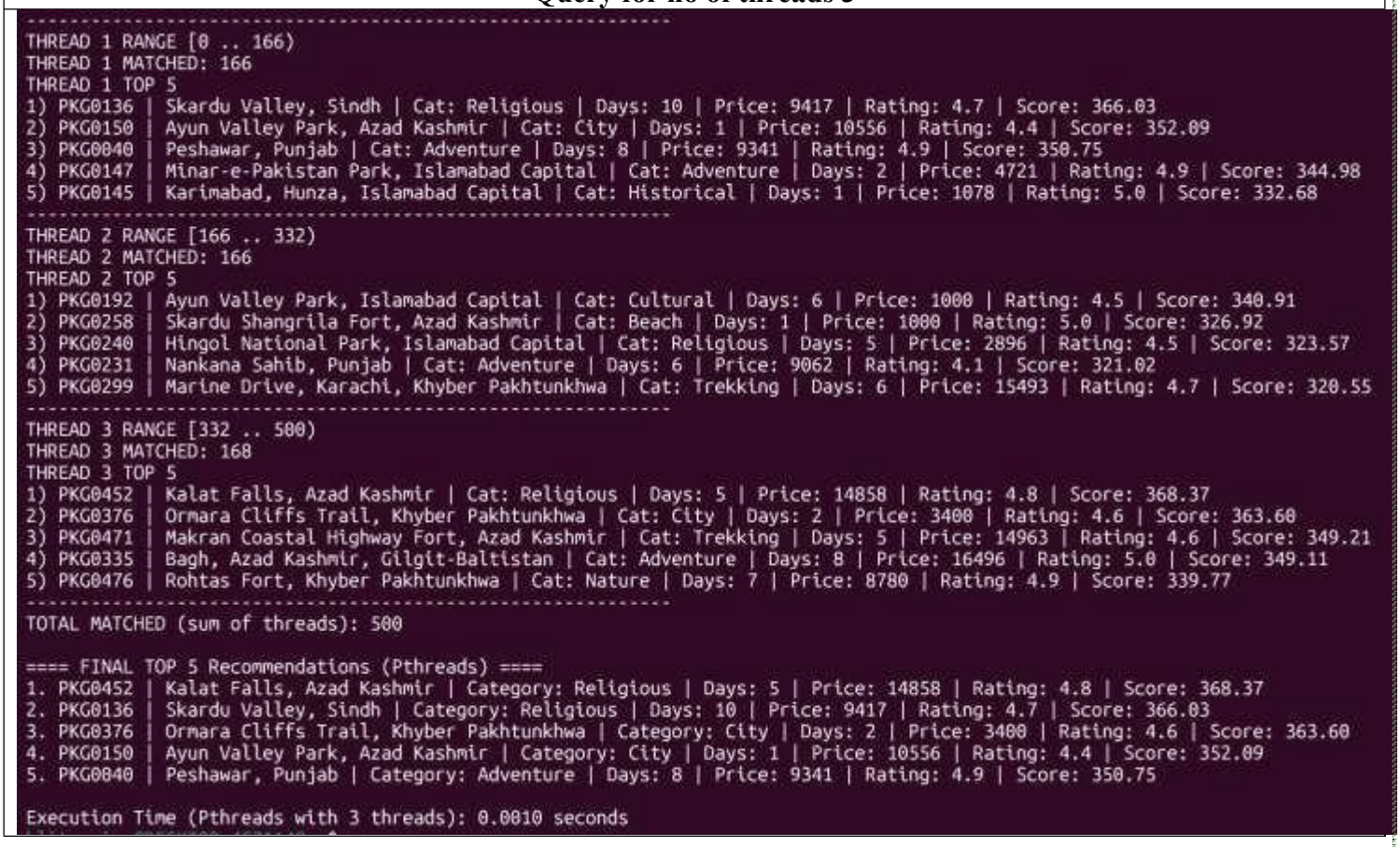
1. Opens dataset file
2. Reads and stores all packages
3. Takes user query
4. Filters packages
5. Calculates score
6. Sorts results
7. Prints top recommendations
8. Shows execution time

This is the brain of the program.

Pthread Implementation (Shared-Memory Multithreading)



Query for no of threads 3



1. **parse_line()**

What it does:

Reads one row from the dataset file and saves its data.

In simple words:

• Takes one line from the file
• Splits it using tab (\t)
• Stores values like package ID, place, province, price, rating, etc.
• Skips the header row

Used for: Loading dataset into memory.

---

2. **reset_query_defaults()**

What it does:

Resets all search filters to default values.

In simple words:

• Clears province and category
• Sets large budget range
• Sets TOPK = 5
• Removes day and rating filters

Used for: Making sure old queries do not affect new ones.

---

3. **is_only_number()**

What it does:

Checks if user input contains only numbers.

In simple words:

• If user types 3, it returns true
• If user types TOPK=3, it returns false

Used for: Treating input like 3 as TOPK = 3.

---

4. **parse_query()**

What it does:

Reads the user's query and sets filters.

In simple words:

• First resets all filters
• If user types only a number → sets TOPK
• Otherwise reads filters like:
o Province
o Category
o Budget
o Days
o Rating
o TopK

Used for: Understanding what the user wants to search.

---

5. **matches_filter()**

What it does:
Checks if a package matches the user query.
In simple words:
• Checks province match
• Checks category match
• Checks budget range
• Checks number of days
• Checks minimum rating

Returns:
• YES (1) → package is valid
• NO (0) → package is rejected

---

6. calculate_score()

What it does:
Gives a score to each package.
In simple words:
• Higher rating → higher score
• More popularity → higher score
• More reviews → higher score
• Price close to budget → extra score
• Days close to required → extra score

Used for: Ranking packages.

---

7. sort_topk()

What it does:
Sorts packages by score (highest first).
In simple words:
• Uses bubble sort
• Best package comes at the top
• Worst package goes down

Used for: Showing best recommendations first.

---

8. process_range() ✓ (THREAD FUNCTION)

What it does:
This is the function that **each thread runs**. It filters and scores packages only in its assigned range.
In simple words:
• Gets thread_id and its range (start to end)
• Loops through packages in that range
• If a package matches filters → calculates score and stores it
• Sorts its own local results
• Saves only its local TOPK results for merging later
• Prints thread results safely using a mutex

Used for: Doing the main work in parallel (each thread handles part of dataset).

---

9. pthread_mutex_lock() / pthread_mutex_unlock() ✓ (PRINT LOCK)

What it does:
Stops threads from printing at the same time.

In simple words:
• Without mutex: outputs mix and become messy
• With mutex: only one thread prints at a time
• Then it releases the lock so other threads can print
Used for: Clean and readable thread output.

---

10. main()

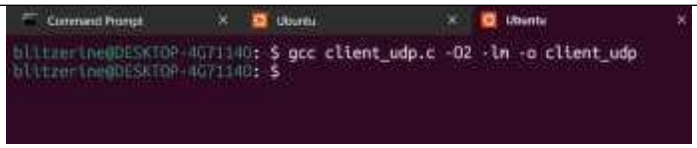
What it does:
Controls the full Pthreads program (loading, threading, merging, final output).
In simple words:

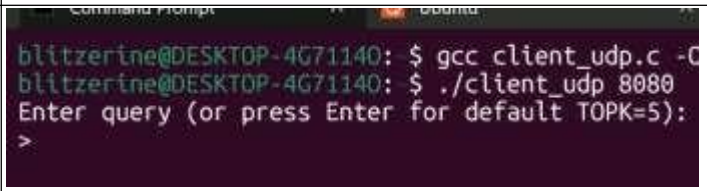
1. Opens dataset file
2. Reads and stores all packages
3. Takes number of threads from user (argv[2])
4. Takes user query (argv OR input)
5. Parses query
6. Divides dataset into ranges for each thread
7. Creates threads (pthread_create)
8. Waits for threads to finish (pthread_join)
9. Collects each thread's TOPK results
10. Merges all thread TOPK results into one list
11. Sorts final merged list
12. Prints FINAL TOPK recommendations
13. Shows execution time (Pthreads)

This is the brain of the Pthreads program.

Serial: One loop processes all packages alone.
Pthreads: Work is split into parts, multiple threads process in parallel, then results are merged.

Client–Server Implementation (UDP-Based Communication)





UDP CLIENT CODE

---

1. is_only_number()

What it does:
Checks if the user input contains only digits.
In simple words:
• If user types 3, it returns true
• If user types TOPK=3, it returns false
Used for: Converting 3 into TOPK=3.

---

2. Command Line Arguments Handling

What it does:
Reads server port and optional server IP from command line.
In simple words:
• Takes server port from user
• Takes server IP if provided
• Uses 127.0.0.1 by default (same machine)
Used for: Knowing where to send the query.

---

3. Socket Creation

What it does:
Creates a UDP socket.

In simple words:
• Uses UDP protocol
• Prepares communication channel
• If socket fails, program stops
Used for: Sending and receiving data.

---

4. Server Address Setup

What it does:
Sets server IP and port in correct format.
In simple words:
• Sets IPv4 family
• Converts port using `htons()`
• Converts IP string into binary form
Used for: Correct UDP communication.

---

5. User Query Input

What it does:
Reads query from the user.
In simple words:
• User can type:
- 3
- or `PROVINCE=Punjab;TOPK=3`
  • Empty input → default `TOPK=5`
  • Converts only-number input into `TOPK=value`

Used for: Getting user search request.

---

6. Sending Query to Server

What it does:
Sends query string to server using UDP.
In simple words:
• Uses `sendto()`
• Sends query to server IP and port
Used for: Requesting results from server.

---

7. Receiving Server Response

What it does:
Receives result from server.
In simple words:
• Waits for server reply
• Stores result in buffer
• Prints received recommendations
Used for: Showing output to user.

---

8. Closing Socket

What it does:
Closes the UDP socket.

In simple words:
• Frees system resources
• Ends client program safely

UDP SERVER CODE

---

1. Global Dataset Arrays

What it does:
Stores dataset information without using structs.
In simple words:
• Stores package ID, place, province
• Stores price, rating, reviews, popularity
• All data kept in arrays
Used for: Holding dataset in memory.

---

2. Query Parameters

What it does:
Stores filters sent by the client.
In simple words:
• Province, category, budget
• Days, rating, TOPK
Used for: Filtering packages.

---

3. parse_line()

What it does:
Reads one row from dataset and saves values.
In simple words:
• Reads one line
• Splits using tab (\t)
• Stores values in arrays
• Skips header row
Used for: Loading dataset.

---

4. parse_query()

What it does:
Reads client query and sets filters.
In simple words:
• Resets filters
• Reads province, category, budget
• Reads days, rating, TOPK
Used for: Understanding client request.

---

5. matches_filter()

What it does:
Checks if a package matches the query.
In simple words:
• Checks province

• Checks category
• Checks budget range
• Checks days and rating

Returns:

• 1 → match
• 0 → reject

---

6. calculate_score()

What it does:
Calculates ranking score of a package.
In simple words:
• Higher rating → higher score
• Higher popularity → higher score
• More reviews → higher score
• Price close to budget → extra score
• Days close to required → extra score

Used for: Ranking packages.

---

7. sort_topk()

What it does:
Sorts packages by score.
In simple words:
• Uses bubble sort
• Highest score comes first

Used for: Getting best recommendations.

---

8. process_query_and_format()

What it does:
Processes query and prepares response text.
In simple words:
• Parses query
• Filters packages
• Calculates scores
• Sorts TOPK results
• Formats results into a string

Used for: Sending readable response to client.

---

9. Dataset Loading (in main)

What it does:
Loads dataset file into memory.
In simple words:
• Opens dataset file
• Reads line by line
• Stores data using parse_line()

Used for: Preparing server data.

---

10. UDP Socket Creation & Binding

What it does:
Creates and binds server socket.
In simple words:
• Opens UDP socket
• Binds to given port
• Ready to receive client queries
Used for: Starting server.

---

11. Receiving Client Queries

What it does:
Receives query from client.
In simple words:
• Waits using `recvfrom()`
• Prints client IP, port, and query
Used for: Accepting client request.

---

12. Sending Response to Client

What it does:
Sends result back to client.
In simple words:
• Calls `process_query_and_format()`
• Sends result using `sendto()`
Used for: Returning recommendations.

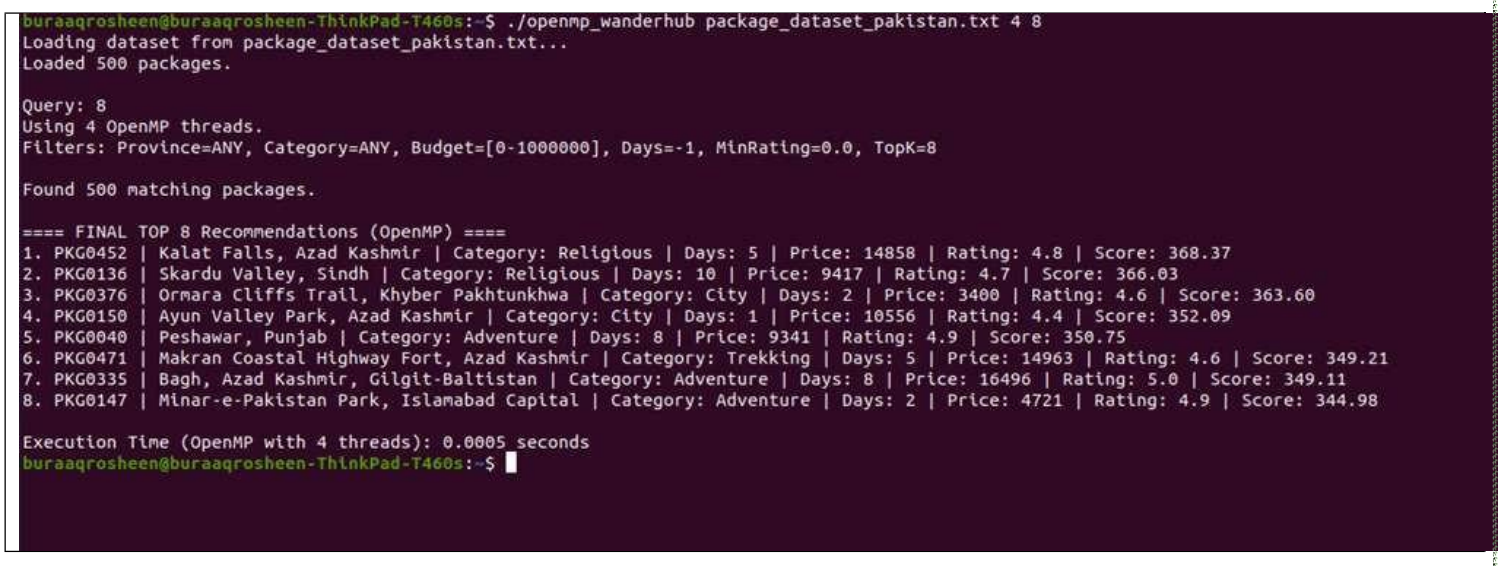
---

13. Continuous Server Loop

What it does:
Keeps server running forever.
In simple words:
• Server never stops
• Handles one query at a time
• Multiple clients supported (UDP)
Used for: Always-on server.

Client sends query → **Server** filters & ranks packages → **Server** sends results → **Client** displays them.

OpenMP Implementation (Shared-Memory Parallelism with Directives)



1. `is_only_number()`

What it does: Checks if input is only digits like `3`.

In simple words:

- `3` → true
- `TOPK=3` → false

Used for: treating `3` as `TOPK=3`.

2. `reset_query_defaults()`

What it does: resets filters to defaults.

In simple words:

- clears province/category
- sets big budget range
- sets TOPK=5
- removes day/rating filter

Used for: making new query clean.

3. `parse_line()`

What it does: reads one dataset row into arrays.

In simple words:

- skips header
- splits by tab `\t`
- stores id/place/province/category/price/rating etc.

Used for: loading dataset in memory.

4. `parse_query()`

What it does: reads query and sets filters.

In simple words:

- resets defaults first
- if only number → set TOPK

• else split by ; and read PROVINCE/CATEGORY/BUDGET/DAYS/MIN_RATING/TOPK

Used for: understanding user input.

5. `matches_filter()`

What it does: checks if package matches filters.

In simple words:

• province match (if given)
• category match (if given)
• budget range match
• days match (if given)
• rating >= min rating

Returns: 1 match, 0 reject.

6. `calculate_score()`

What it does: calculates score for ranking.

In simple words:

• rating increases score
• popularity increases score
• more reviews increases score
• budget closeness adds extra score
• duration closeness adds extra score

Used for: ranking top packages.

7. `sort_topk()`

What it does: sorts scores high to low.

In simple words:

• bubble sort
• best score comes first

Used for: showing best results on top.

8. `main()` (OpenMP part)

What it does: runs whole OpenMP program.

In simple words:

1. loads dataset
2. reads and parses query
3. sets number of threads
4. **parallel loop** filters + scores packages
5. stores matching results
6. sorts and prints TOPK
7. prints execution time

Used for: shared-memory parallel execution on one machine.

MPI Implementation (Distributed-Memory Message Passing)

```
buraaqrosheen@buraaqrosheen-ThinkPad-T460s:~$ mpirun -np 2 ./mpi_wanderhub package_dataset_pakistan.txt
Loading dataset from package_dataset_pakistan.txt...
Loaded 500 packages.

Enter query (example: TOPK=3 or PROVINCE=Punjab;CATEGORY=Nature;TOPK=3)
You can also just type: 3  (means TOPK=3)
> 6

Query: 6
Using 2 MPI processes.
Filters: Province=ANY, Category=ANY, Budget=[0-1000000], Days=-1, MinRating=0.0, TopK=6

------------------------------------------------------------
TOTAL MATCHED (sum of ranks): 500

==== FINAL TOP 6 Recommendations (MPI/OpenMPI) ====
1. PKG0452 | Kalat Falls, Azad Kashmir | Category: Religious | Days: 5 | Price: 14858 | Rating: 4.8 | Score: 368.37
2. PKG0136 | Skardu Valley, Sindh | Category: Religious | Days: 10 | Price: 9417 | Rating: 4.7 | Score: 366.03
3. PKG0376 | Ormara Cliffs Trail, Khyber Pakhtunkhwa | Category: City | Days: 2 | Price: 3400 | Rating: 4.6 | Score: 363.60
4. PKG0150 | Ayun Valley Park, Azad Kashmir | Category: City | Days: 1 | Price: 10556 | Rating: 4.4 | Score: 352.09
5. PKG0040 | Peshawar, Punjab | Category: Adventure | Days: 8 | Price: 9341 | Rating: 4.9 | Score: 350.75
6. PKG0471 | Makran Coastal Highway Fort, Azad Kashmir | Category: Trekking | Days: 5 | Price: 14963 | Rating: 4.6 | Score: 349.21

Execution Time (MPI with 2 processes): 0.0001 seconds
buraaqrosheen@buraaqrosheen-ThinkPad-T460s:~$
```

`is_only_number()`

What it does:

Checks if the user input is only digits.

In simple words:

- If input is `3` → it returns true
- If input is `TOPK=3` → it returns false

Used for: Allowing user to type only a number to set TOPK.

`reset_query_defaults()`

What it does:

Resets all query filters to default values.

In simple words:

- Clears province and category filters
- Sets budget range wide (0 to 1000000)
- Removes days filter
- Sets minimum rating to 0
- Sets TOPK to 5

Used for: Making sure every new query starts clean.

`parse_line()`

What it does:

Reads one row from the dataset file and stores it in arrays.

In simple words:

- Skips header line (`package_id`)
- Splits line using tab `\t`
- Saves values into arrays like:
  - package id
  - place name

- province
- category
- days, price, rating, reviews, popularity

Used for: Loading the dataset into memory (only rank 0 reads file).

`parse_query()`

What it does:
Reads the query string and sets filtering variables.

In simple words:
• First resets defaults
• If query is just a number like `3` → sets `query_topk = 3`
• Otherwise splits query by `;`
• Reads each filter like:

- `PROVINCE=...`
- `CATEGORY=...`
- `BUDGET_MIN=...`
- `BUDGET_MAX=...`
- `DAYS=...`
- `MIN_RATING=...`
- `TOPK=...`

Used for: Understanding what user wants in search.

`matches_filter()`

What it does:
Checks if a package matches the user query filters.

In simple words:
• If province filter exists → must match
• If category filter exists → must match
• Price must be in budget range
• If days filter exists → duration must match
• Rating must be >= min rating

Returns:
• 1 → package is allowed
• 0 → package is rejected

`calculate_score()`

What it does:
Calculates a score to rank packages.

In simple words:
• Higher rating → higher score
• Higher popularity → higher score
• More reviews → higher score using log(reviews+1)
• If user set budget max → adds bonus for price close to budget
• If user set days → adds bonus for duration close to days

Used for: Ranking packages from best to worst.

`sort_topk()`

What it does:
Sorts packages by score (highest first).

In simple words:
• Uses bubble sort

• Swaps items if score is smaller
• After sorting, best score is at index 0
Used for: Getting top recommendations.

`bcast_dataset()`
What it does:
Sends dataset from **rank 0 (master)** to all ranks using MPI broadcast.
In simple words:
• Rank 0 loads dataset from file
• Then it broadcasts:
- total number of packages
- all arrays (ids, names, provinces, categories, prices, ratings, etc.)
  • Every rank gets the same dataset in its own memory

Used for: Making sure all MPI processes have the dataset.

`main()`
What it does:
Controls the full MPI program from start to end.

1. Starts MPI (`MPI_Init`)
2. Gets rank id and number of processes (`rank`, `world`)
3. Rank 0 reads dataset file
4. Dataset is broadcast to all ranks (`bcast_dataset`)
5. Rank 0 reads query, broadcasts query to all ranks (`MPI_Bcast`)
6. Each rank processes only its part of dataset:
   • computes its start/end range
   • applies filters + computes scores
   • sorts local results and keeps local TOPK
7. MPI collects results on rank 0:
   • `MPI_Gather` gets match counts
   • `MPI_Gather` gets topk counts
   • `MPI_Gatherv` collects indices + scores
8. Rank 0 merges, sorts again, prints final TOPK
9. Prints total execution time using `MPI_Wtime()`
10. Ends MPI (`MPI_Finalize`)

Used for: Running the recommender in distributed parallel way using MPI.

Comparison of Execution Models

Feature	Serial	Pthreads	OpenMPI (MPI)	Client–Server (UDP)
Execution Type	Sequential	Parallel	Parallel	Distributed
Processing Style	Single thread	Multiple threads	Multiple processes	Client sends, server processes
Memory Model	Single memory	Shared memory	Distributed memory	Separate memory (network)
Runs On	Single CPU	Single machine	Multiple machines or cores	Multiple machines
Communication	None	Shared variables	Message passing	Network messages
Speed	Slowest	Faster	Very fast (scalable)	Depends on network
Complexity	Very simple	Medium	High	Medium
Synchronization	Not required	Mutex/locks	MPI messages	Network protocol
Scalability	Very poor	Limited	Excellent	Good
Fault Tolerance	None	Low	Medium	High
Best Use Case	Small data	Multi-core systems	Clusters / HPC	Remote querying

Advantages Table

Model	Advantages
Serial	• Very easy to write and debug• No synchronization issues• Best for learning and small datasets
Pthreads	• Faster than serial• Uses all CPU cores• No network overhead• Efficient shared memory access
OpenMPI (MPI)	• Highly scalable• Can run on multiple machines• Best for large datasets• Industry standard for HPC
Client–Server	• Remote access supported• Multiple clients supported• Server always running• Platform independent

Disadvantages Table

Model	Disadvantages
Serial	• Very slow for large datasets• Wastes multi-core CPUs
Pthreads	• Shared memory issues• Race conditions possible• Harder to debug
OpenMPI (MPI)	• Complex code• Requires MPI setup• Message passing overhead
Client–Server	• Network delay• UDP is unreliable• Server can be bottleneck

When to Use Which?

Situation	Best Choice
Learning / Testing	Serial
Multi-core PC	Pthreads
Cluster / Supercomputer	OpenMPI
Remote Query System	Client–Server

Time Comparison Table

Model	Observed Execution Time	Relative Speed	Explanation (Based on Results)
Serial (Sequential)	**Highest (Slowest)**	⭐	Uses **single CPU core**. All filtering, scoring, and sorting are done **one-by-one**, so total time increases linearly with dataset size.
Pthreads (3 Threads)	**Low**	⭐ ⭐ ⭐	Dataset is split among **multiple threads**. Filtering and scoring run **in parallel**, reducing execution time compared to serial. Some overhead due to thread creation and mutex locks.
OpenMP	**Lower than Pthreads**	⭐ ⭐ ⭐ ⭐	Uses compiler-level parallelism with **less overhead**. Automatic work-sharing results in better CPU utilization than manual thread management.
MPI (OpenMPI)	**Very Low (Fastest)**	⭐ ⭐ ⭐ ⭐ ⭐	Work is divided among **multiple processes**. Each process computes independently, then results are merged. Scales best for large datasets and clusters.
Client–Server (UDP)	**Medium**	⭐ ⭐	Server computation is fast, but **network communication (send/receive)** adds delay, increasing total response time compared to pure local execution.