

Understanding Parallelism & Concurrency

(informal introduction)

Mahmoud Parsian

Ph.D. in Computer Science

Concurrency & Parallelism

(informal definition):

- The fact of two or more events or circumstances happening or existing at the same time.
- The ability to execute more than one program or task simultaneously.
- Example: "a high level of concurrency is crucial to good performance in a multiuser database system"

Parallelism

Parallelism is basically a type of computation in which many computations or operations are carried out in parallel.

Sequential tasks:

Task-1: 15 minutes

Task-2: 20 minutes

Task-3: 25 minutes

Task-4: 20 minutes

=====

Total Elapsed time: 80 minutes

(assuming that there is no dependencies between tasks)

Parallel tasks:

Task-1, Task-2, Task-3, Task-4

=====

Total Elapsed time: 25 minutes

Therefore, Parallelism improves execution time.

Parallelism

What if Task-3 and Task-4 depends on output of Task-1 and Task-2

Sequential tasks:

Task-1: 15 minutes

Task-2: 20 minutes

Task-3: 25 minutes

Task-4: 20 minutes

=====

Total Elapsed time: 80 minutes

(assuming that there is no dependencies between tasks)

Dependencies can be bottlenecks!!!

Parallel tasks:

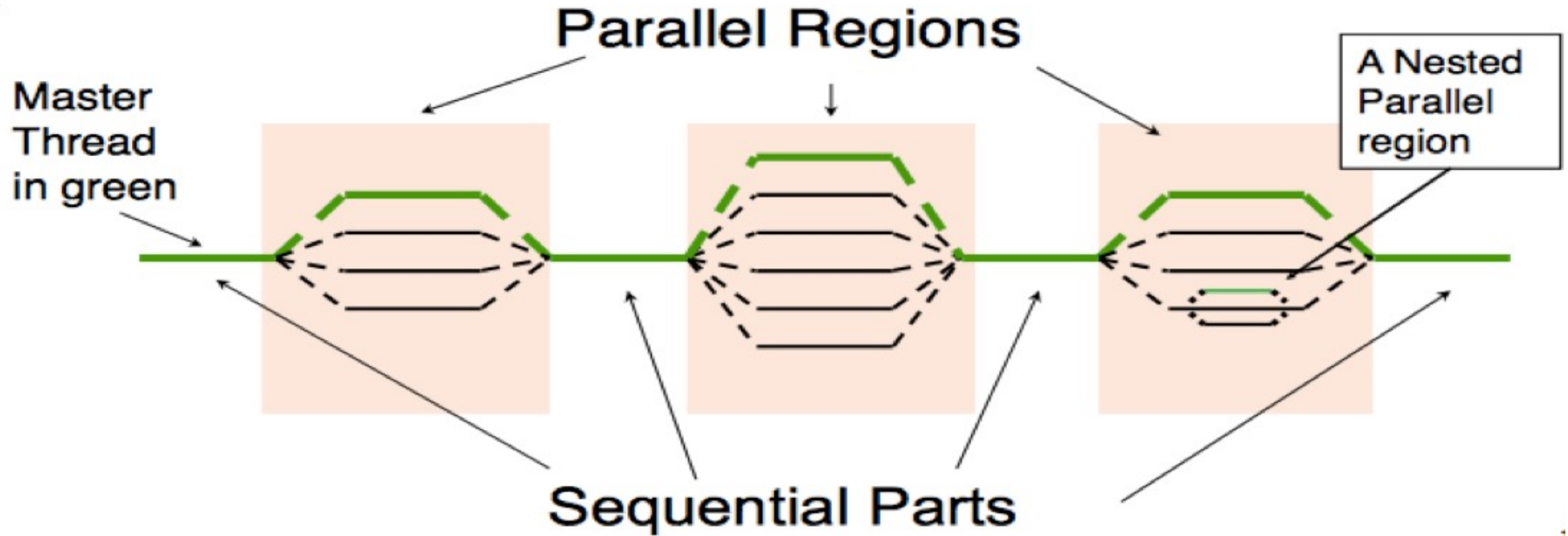
Iteration-1: Task-1, Task-2 (duration: 20 mins)

Iteration-2: Task-3, Task-4 (duration: 25 mins)

=====

Total Elapsed time: 45 minutes

Parallelism



Birthday Example

- Alex wants to throw a big birthday party.
- He has a list of 1000 items to buy from a Safeway (grocery store).
- Items to buy:
 - item-1: ice-cream
 - item-2: chips
 - item-3: oranges
 - item-4: cup cakes
 - ...
 - item-1000: grapes
- How long will it take for Alex to buy all these 1000 items
 - assuming that he is the only person in grocery store?

What is our Data?

ID	Item to buy
1	ice-cream
2	chips
3	oranges
4	Cup-cakes
...	...
1000	grapes

Birthday Example ...

- How long will it take for Alex to buy all these 1000 items (assuming that he is the only person in grocery store)?
- It is estimated that on average he needs 14 seconds to find the item and put in the shopping cart.
- Therefore, he will spend

$$1000 \times 14 = 14,000 \text{ seconds} = 234 \text{ minutes}$$

BUT, Alex is busy and can not spend **234 minutes** for shopping.

What to do?

Processing Data? One Executor: Alex
Buy item-1, then item-2, then item-3, ...
All is done in sequence: No Parallelism yet.

ID	Item to buy
1	ice-cream
2	chips
3	oranges
4	Cup-cakes
...	...
1000	grapes

Birthday Example ...

BUT, Alex is **busy** and can not spend **234 minutes** for shopping.

What to do?

- Alex calls 10 of his friends (F1, F2, ..., F10): they will do shopping and deliver to Alex.
- Therefore, Each friend will buy 100 items
- Total time elapsed to buy all items:
 $100 \times 14 = 1400 \text{ seconds} = 24 \text{ minutes}$
- **THIS IS a BIG IMPROVEMENT: 234 minutes reduced to 24 minutes**
- **Alex is still not satisfied with this plan.**

Processing Data? 10 parallel executors

Each executor operates in parallel & independently

Executor-1: F1

ID	Item to buy
1	ice-cream
2	chips
3	oranges
4	Cup-cakes
...	...
100	grapes

Executor-1: F2

ID	Item to buy
101	...
102	...
103	...
104	...
...	...
200	grapes

...

Executor-10: F-10

ID	Item to buy
901	...
902	...
903	...
904	...
...	...
1000	grapes

Birthday Example ...

BUT, Alex is **busy** and can not spend 24 minutes for shopping.

What to do?

- Alex calls 100 of his friends (F1, F2, ..., F100):
they will do shopping and deliver to Alex.
- Therefore, Each friend will buy 10 items
- Total time elapsed to buy all items:
 $10 \times 14 = 140$ seconds = about 3 minutes
- THIS IS a BIG IMPROVEMENT: **234 minutes** reduced to **3 minutes**
- Alex is still not satisfied with this plan.

Processing Data? 100 parallel executors

Each executor operates in parallel & independently

Executor-1: F1

ID	Item to buy
1	ice-cream
2	chips
3	oranges
4	Cup-cakes
...	...
10	...

Executor-2: F2

ID	Item to buy
11	...
12	...
13	...
14	...
...	...
20	...

...

Executor-100: F-100

ID	Item to buy
991	...
992	...
993	...
994	...
...	...
1000	grapes

Birthday Example ...

BUT, Alex is **busy** and can not even spend 3 minutes for shopping.

What to do?

- Alex calls 1000 of his friends (F1, F2, ..., F1000): they will do shopping and deliver to Alex.
- Therefore, Each friend will buy 1 single item
- Total time elapsed to buy all items:
 $1 \times 14 = 14$ seconds
- THIS IS a HUGE IMPROVEMENT:
234 minutes reduced to 14 seconds
- Alex is satisfied with this plan.

Processing Data? 1000 parallel executors

Each executor operates in parallel & independently

Executor-1: F1

ID	Item to buy
1	ice-cream

Executor-2: F2

ID	Item to buy
2	chips

...

Executor-1000: F-1000

ID	Item to buy
1000	grapes

How do we write parallel programs?

- **Task parallelism**

- Partition various tasks carried out solving the problem among the cores.

- **Data parallelism**

- Partition the data used in solving the problem among the cores.
- Each core carries out similar operations on it's part of the data.

Data parallelism

- Let your data has 200,000,000,000 data points
- Partition your data into 100,000 chunks:
 - Number of partitions: **100,000**
 - Number of records per partition: **2000,000**
 - **100,000 x 2000,000 = 200,000,000,000**
- Assume you want to execute **map(function)** on (a transformation function) each record and create a new record
- With these partitioning in place, The fastest way to execute **map(function)** will be to have **100,000** mappers (mappers are transformations, which execute in parallel)
- What happens if we have only 1000 mappers?

Data parallelism

- What happens if we have only 1000 mappers?
- First, we assign 1000 of these partitions to 1000 mappers (**each mapper gets a single partition, which has 2,000,000 records**)
- Once a mapper completes its task, we assign another partition to that mapper
- This iteration continues until we exhaust all 100,000 partitions
- **At most 1000 mappers are executing at a single point of time.**
- **The more mappers we have:
The more we execute faster**

Parallelism requires Coordination

- Executors usually need to coordinate their work.
- **Communication** – one or more Executors send their current partial results to another core.
- **Load balancing** – share the work evenly among the Executors so that one is not heavily loaded.
- **Synchronization** – because each Executors works at its own pace, make sure Executors do not get too far ahead of the rest.

NOTE: If you use **Spark or MapReduce**, then all of these are done automagically for you!!!

Partitioner

- A partitioner partitions the input into chunks.
- If input has 200,000,000,000 records and we partition this input into 200,000 chunks, then we have:
 - Number of partitions: 200,000
 - Size of each partition: 1000,000 records
 - $200,000,000,000 = 200,000 \times 1,000,000$

Typically, a chunk (1,000,000 records) becomes a unit of parallelism.

Chunk = Partition

Benefits of Parallel Computing

- **Parallel computing models the real world.** The world around us isn't serial and sequential: many things happen at the same time
- **Saves time.** Serial/sequential computing forces fast processors to do things inefficiently.
- **Saves money.** By saving time, parallel computing makes things cheaper and faster
- **Solve more complex or larger problems** by partitioning them into smaller problems
- **Solve Larger Problems in a short point of time.**
- **Leverage all available resources.**

References

- Introduction to Parallel Computing
 - <https://www.geeksforgeeks.org/introduction-to-parallel-computing/>
- Introduction: Parallelism = Opportunities + Challenges
 - <https://courses.cs.washington.edu/courses/csep524/07sp/poppChaper1.pdf>