MapReduce: Programming Model

MapReduce is a programming model and an associated implementation for processing and generating **big data** sets with a **parallel**, **distributed** algorithm on a **cluster**. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.

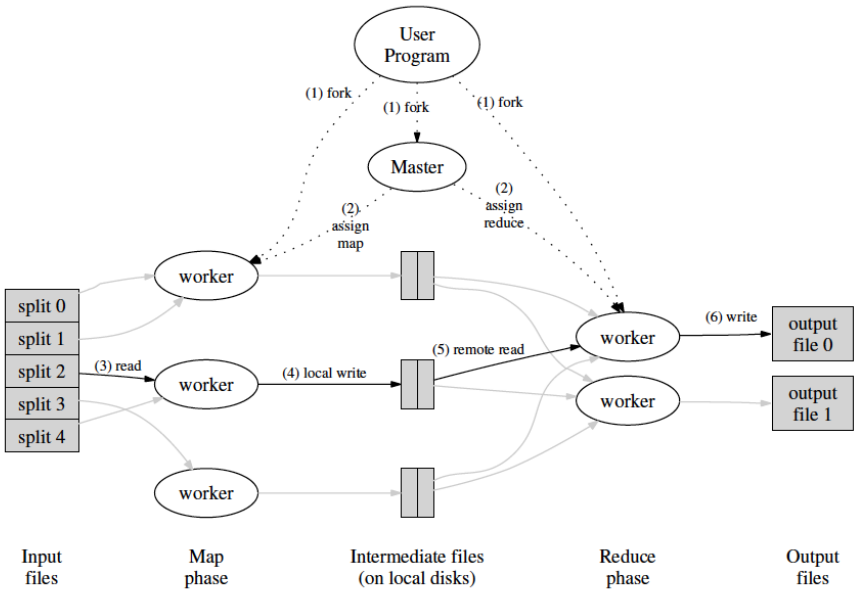


Figure 1: Execution overview

When do we use OpenMP, MPI and, Hadoop MapReduce, and why?

OpenMP: If you want to introduce shared memory parallelism to your code, then OpenMP is an efficient directive based library that you could use. For example let's say you have a for loop that you want to parallelize. Using OpenMP, you can put a directive as #pragma parallel for before the for loop and it'll split the loop into multiple threads, so each of them would handle a chunk of the loop's iterations.

There's more to OpenMP and it's a very neat and powerful library to do what you otherwise might end up writing manual thread code. Just keep in mind, OpenMP can't do 'magics' for you, so you need to be careful how you use it to avoid performance and correctness pitfalls.

MPI: Message Passing Interface is a distributed memory parallel model implementation, typically used to develop parallel scientific applications. The reason I say scientific applications is because they are usually tightly synchronous code and well load balanced. Why this is important is that having stragglers makes MPI program inefficient.

Anyway, performance aside, MPI can be used to develop pretty much any parallel code that runs over multiple machines. Note, if you want, you can mix MPI with OpenMP to use threads within machines. This is called hybrid programing.

Another, important thing to note is that although it's possible to implement dataflow styled parallel programs using MPI, it might not be the best choice.

Hadoop MapReduce: Now, think of Hadoop as giving you two constructs that you can apply over your large data; map and reduce. The idea is data is distributed, typically using HDFS (The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware.), and you want to apply some operation for each data element. Then you can do some reduction over the results. While this pattern may seem simple it is possible to chain maps and reduces to design complex problems.

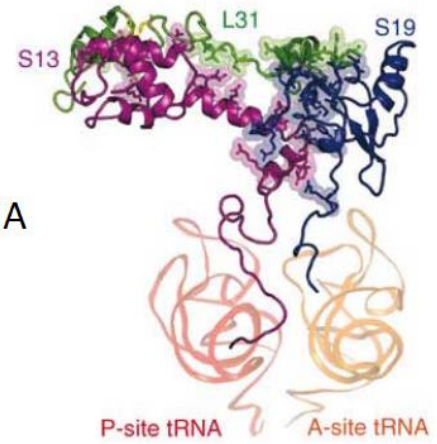
Now, when would you want to use Hadoop and maybe not MPI? Typically, if you have terabytes of data that you want to do ETL (extract, transform and load) like operations. Fault tolerance is one of the most important features when using Hadoop. Note, while MapReduce can be used to write tightly coupled scientific applications as well, performance wise it's generally not a good idea. There have been improvements though.

Drug Design and DNA: Problem overview

How do pharmaceutical companies design the medicines we use?

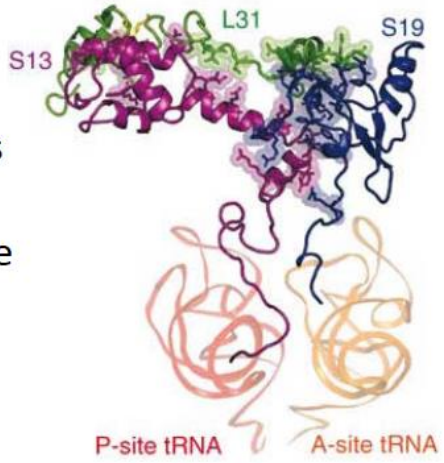
How medicines work

- Our **DNA** is like a book of recipes
- Instead of food, DNA contains the instructions for making **proteins** in our bodies

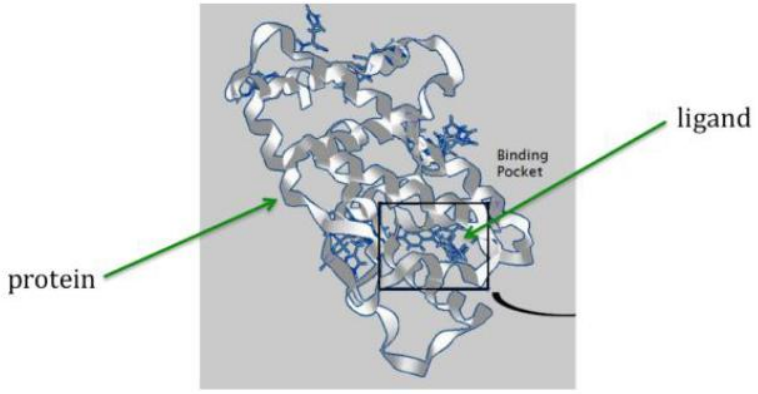


How medicines work

- A **protein's shape** determines the function it performs in a person's body
- To design a drug, we can **find *ligands* (new pieces)** to change a protein's shape



How medicines work



Strategy for drug design software

1. **Generate ligands** to try for particular protein
   - Some ligands will fit, some won't
2. Compute a **score for each ligand** that simulates how well it will:
   - fit that protein; and
   - produce a desired shape change
3. Identify the **highest scoring ligands** for actual synthesis (production) and testing

Drug design exemplar code

A program *structured like* drug design software

1. **Generate ligands** to try for particular protein
   - Random character *strings* of random lengths
2. Compute a **score for each ligand**
   - Compare for **maximum match** with string representing a protein
   - Insertions and deletions allowed
3. Identify the **highest scoring ligands**

l c a c x e t q v i v g
c x t b c r v

Drug design exemplar code

A program *structured like* drug design software

1. **Generate ligands** to try for particular protein
   - Fast
2. Compute a **score for each ligand**
   - Takes a long time
   - **Parallelize** by using **multiple computation threads** for different ligands
3. Identify the **highest scoring ligands**
   - Fast – just sort and find maximum

l c a c x e t q v i v g
c x t b c r v

Drug design exemplar code

Command-line arguments:

./drugdesign-static ***threads maxlen count***

- ***threads*** is number of simultaneous threads
- ***maxlen*** is *maximum* length of a ligand
  - *Each ligand* has random length up to this max
- ***count*** is number of ligands to score