Partitioning and Divide-and-Conquer Strategies

The idea of Divide and Conquer is nothing new in Software Development, just because of Parallel Programming. But while Divide and Conquer for non-parallel computing is about splitting bigger tasks into smaller ones so they are easier to solve, in parallel programming we want to split bigger tasks into smaller ones, so we can distribute them among different nodes (computers, processors, processes …). For splitting them, we first need…

Partitioning

There exist 2 different partitioning strategies:

1. **Data partitioning** – divide the data and perform the same task on different data. Afterwards collect the results and merge them.
2. **Domain decomposition** – partition some function or functionality of a program. Perform different parts of a function on each node. For this, the parts must not depend upon each other, which is, why this strategy is of course harder than data partitioning.

This may sound easy at first, but you need to realize, that **not everything** can be partitioned.

> *Small example, you can divide:* $\sqrt{x * y}$ *into* $\sqrt{x} * \sqrt{y}$.
> *But you cannot divide* $\sqrt{x + y}$ *into* $\sqrt{x} + \sqrt{y}$.

Distributing Work

If you do Data Partitioning, you might want to use shared memory and just send the working position to every process. Since this is not possible on clusters, grids… you need to send the data on which the node will work as well. There are several ways to do it:

1. The easiest: send to each node with unicast the data it should process. Costs a lot of resources, very slowly -> **DON'T TRY THIS AT HOME**
2. Broadcast / Multicast the data to all nodes and these know, based on their ID or other information, which part of the data they need to process.
3. Similar to the first one but with more thought inside, create a tree:
    a. Node splits the data into X equal parts (if x = 2 you get a binary tree)
    b. Node shares the parts in 2 different ways:
        i. with X different processes (== This node has nothing to do until it's children finish computing)
        ii. with X-1 process (== This node keeps a part of the data and performs with it the next step)
    c. Once the node has the data, it can
        i. Go back to (a), if the data should be partitioned again
        ii. Process the data and return a result

The last step divides the most of work between many nodes, since not only the processing of the data, but also partitioning and merging is divided between nodes. And you end up with a nice tree:

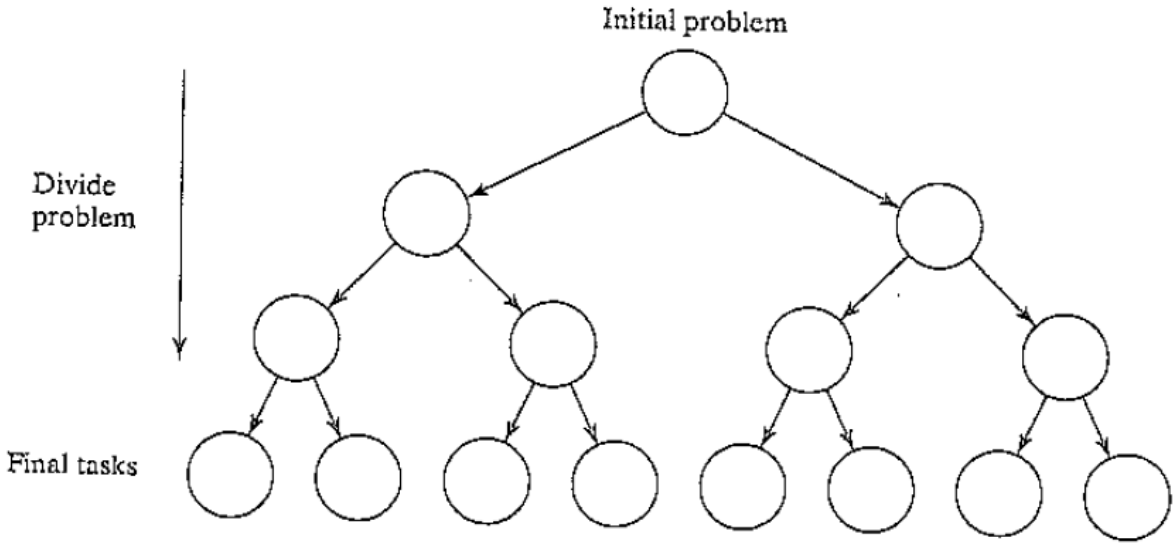
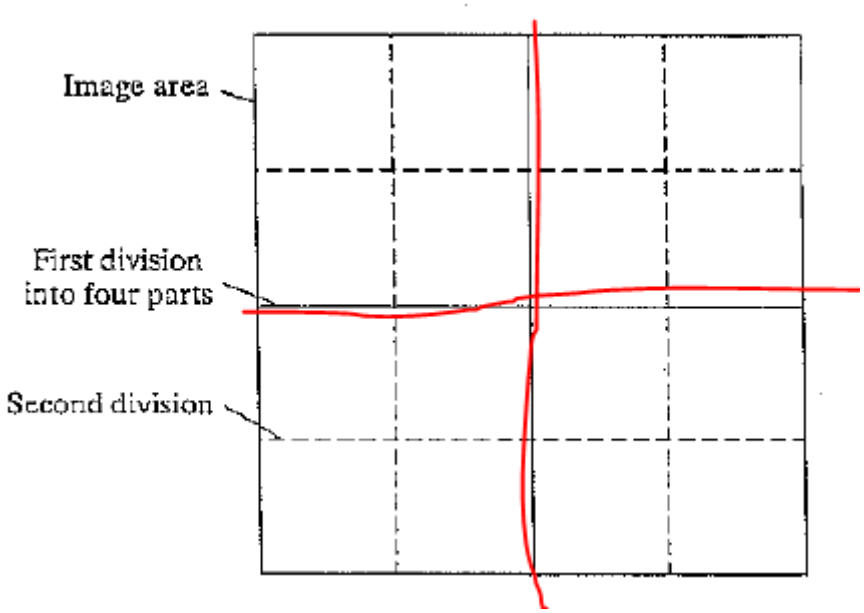


Figure 4.2 Tree construction.

How / why would I split data in more than 2 parts?

I guess it's common to think of data as lists or arrays and it's weird to split these into 3 or more parts (see quicksort, merge sort…). But if you have an image, map or any other kind of 2D data (2D Array) it's easier to split them In 4 equally big parts (see picture) which would result in 4 children of a node. And since dividing 2D creates 4 parts (2^2) dividing 3D models would create 8 children (2^3) and so on …



Merging results

Once the task has been divided and conquered, you can merge the results, which is also very easy if you split the tasks into a tree formation:

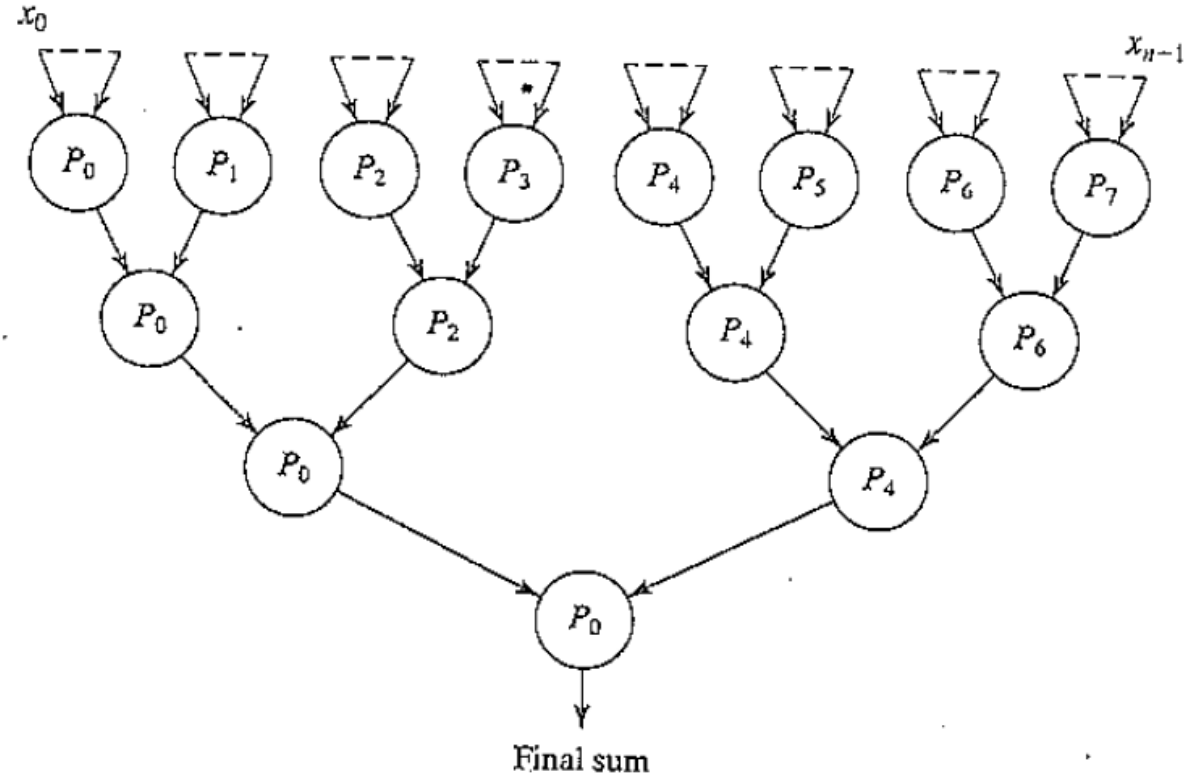


Figure 4.4 Partial summation.

Examples how to use Partitioning and Divide and Conquer

- Computing the Sum, Min, Max of a list (only one value is the result of the work of each node)
- Sorting (Merge sort is the best example here, and also my demo application)
- Numerical Integration
    - Computing integrals in a computer is not really an easy job. Numerical Integration tries to get the result by creating many thin squares or trapezes which assimilate the function. For this, each node could get a small part of the function and compute the value.

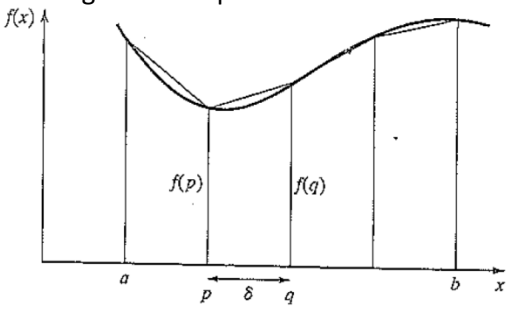


Figure 4.15 Numerical integration using the trapezoidal method.

- N-Body-Problem
    - If 3 or more bodies are in the space and have a mass == gravity, it's not possible for physicists to compute this mathematically. So a Divide and Conquer algorithm can be applied in a computer model to compute the positions of the bodies step by step. Where the bodies or clusters of bodies would be divided within the nodes. This is very complex, since the position of all bodies changes after each computation round and the partitioning needs to be done again.