Designing Parallel Algorithms

Octavio Palomeque Gasperin
ID: 151884
octavio.palomequegn@udlap.mx

13 February 2019

1 Abstract

Most programming problems have several parallel solutions. The fourth stages that are the ones that the text propose as they are the most important ones are:
• Partitioning: The computation that is to be performed and the data operated on by this computation are decomposed into small tasks. Practical issues such as the number of processors in the target computer are ignored, and attention is focused on recognizing opportunities for parallel execution.
• Communication: The communication is required to coordinate task execution in determined, and appropriate communication structures and algorithms are defined.
• Agglomeration: The task and communication structures defined in the first two stages of a design are evaluated with respect to performance requirements and implementation costs.
• Mapping: Each task is assigned to a processor in a manner that attempts to satisfy the competing goals of maximizing processors utilization and minimizing communication costs.

The outcome of this design process can be a program that creates and destroy tasks dynamically, using load-balancing techniques to control the mapping of tasks to processors. Algorithm design is presented here as a sequential activity in practice, however, it is highly parallel process, with many concerns being considered simultaneously. In the next sections we could have a more detailed exploration about the first one called partitioning

2 Partitioning

The partitioning stage of a design is intended to expose opportunities for parallel execution. The main focus is to define a large number of tasks that can be executed separated in what could be called something fine-grained. A fine-grained problem has the possibility to be more flexible when is intended to apply

parallelism. In later stages as the communication stage, the target arquitecture or software engineering will require of other options in the selection of the partitioning. Then we will revisit the original partition and agglomeration task to increase their size or granularity.

A good partition divides into small pieces both the computation associated with a problem and the requirements of data on which this computation operates. When we are designing the way of partition the first thing that we have to define is the data associated with a problem and then determine an appropriate partition for the data, and finally work out how to associate computation with data. This is called domain decomposition.

2.1 Domain Decomposition

In the domain composition approach to problem partitioning, we seek first to decompose the data associated with a problem. If possible, we divide these data into small pieces of approximately equal size.

The data that are decomposed may be the input to the program, the output computed by the program or intermediate values maintained by the program. Different partitions are possible, based on different data structures. Different phases of the computation may operate in different data structures or demand different decompositions and parallel algorithms developed.

2.2 Functional Decomposition

Functional decompisition represents a different and complementary way of thinking about problems. In this approach the initial focus is in the computation that is to be performed rather than on the data that is going to be computed. If we are successful in dividing this computation into disjoint tasks, we proceed to examine the data requirements of this tasks.
While domain decomposition forms the foundation for most parallel algorithms, functional decomposition is valuable as a different way of thinking about problems. For this reason alone, it should be considered when exploring possible parallel algorithms.

2.3 Partitioning Design Checklist

Finally, the partitioning phase of a design should produce one more possible decomposition of a problem. Before proceeding to evaluate communication requirements, we use the following checklist to ensure that the design has no obvious flaws. If you follow this steps then you are ready to go with communication: • Does your partition define at least an order of magnitude more tasks than there are processors in your target computer? If not, you have little flexibility in subsequent design stages
• Does your partition avoid redundant computation and storage requirements? If not, the resulting algorithm may not be scalable to deal with large problems.
• Are tasks of comparable size? If not, it may be hard to allocate each processor

equal amounts of work.

- Does the number of tasks scale with problem size? Ideally, an increase in problem size should increase the number of tasks rather than the size of individual tasks. If this is not the case, your parallel algorithm may not be able to solve larger problems when more processors are available.
- Have you identified several alternative partitions? You can maximize flexibility in subsequent design stages by considering alternatives now. Remember to investigate both domain and functional decomposition.