Parallel and Distributed Algorithms and Programs

Based on the lectures of Anne Benoit
Notes written by Hugo Salou

September 8, 2025

Contents

This course will mostly be on parallel algorithms, but another course, next semester, named *Distributed Systems* will give more knowledge on distribute algorithms/systems. The class is on Tuesday at 1:30PM–3:30PM in Amphi B.

Tutorials will be on Wednesday at 10:15AM–12:15AM in rooms 103 and 105. Some tutorials will require a laptop. Tutorials will be done

There will be a midterm exam on November 4th, a (solo) programming project due December 5th and a final exam on the week of December 15th.

1 Introduction

Parallel computing has been developed to solve many computational problems. Platforms can represent multicore chips, cluster of servers, grids, clouds (they provide services, virtualized resources), fog (some services are precessed at the "edge" of cloud, smart devices). One question we can ask ourselves is: what to optimize? A few examples include:

- total computation time (most parallelism);
- cost of infrastructure;
- total number of operations (parallelism overheard);
- to be able to solve larger problems;
- minimize energy/power (use renewable energy?).

2 A Toy Model: Sorting Networks.

Here we use *comparators*: a very small processor able to, given a and b, sort the two numbers. Can we organize and connect comparators in order to sort a sequence of numbers?

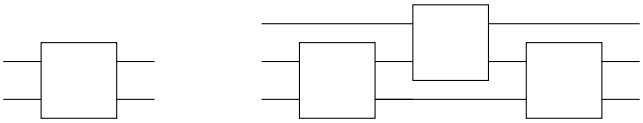

Figure 2.1 | *Sorting networks for two, three and four numbers*

Here we want to optimize in two ways: minimize the number of steps, and minimize the number of comparators.

2.1 Odd-Even merge sort.

The Odd-Even merge sort is a parallel algorithm very similar to merge sort: we start by sorting two halves and then merging the two. In this section, we will consider an input of size $n = 2^m$.

2.1.1 Odd-Even merging network.

Given a sequence $c = \langle c_1, c_2, \ldots, c_n \rangle$, we want to compute a sequence written $\mathsf{SORT}(\langle c_1, c_2, \ldots, c_n \rangle)$ which corresponds to the sorted sequence of the c_i's. If a sequence c is sorted, we write $\mathsf{SORTED}(\langle c_1, \ldots, c_n \rangle)$.

An important part of the merge sort is the following:

if $\mathsf{SORTED}(\langle a_1, \ldots, a_n \rangle)$ and $\mathsf{SORTED}(\langle b_1, \ldots, b_n \rangle)$ then

$$\mathsf{MERGE}(\langle a_1, \ldots, a_n \rangle, \langle b_1, \ldots, b_n \rangle) = \mathsf{SORT}(\langle a_1, \ldots, a_n, b_1, \ldots, b_n \rangle).$$

Let MERGE_m be a network merging two sequences of size $n = 2^m$.