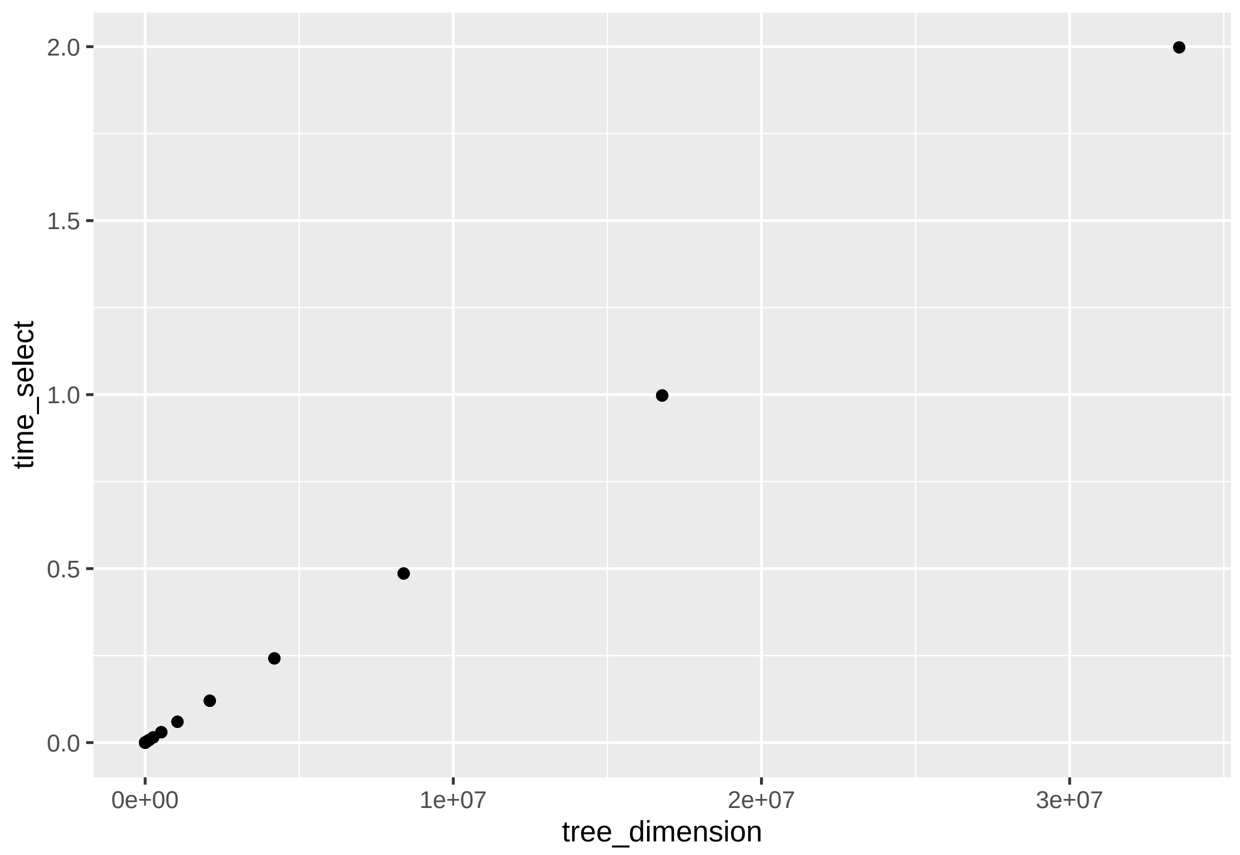
Foundations of High Performance Computing

Assignment II

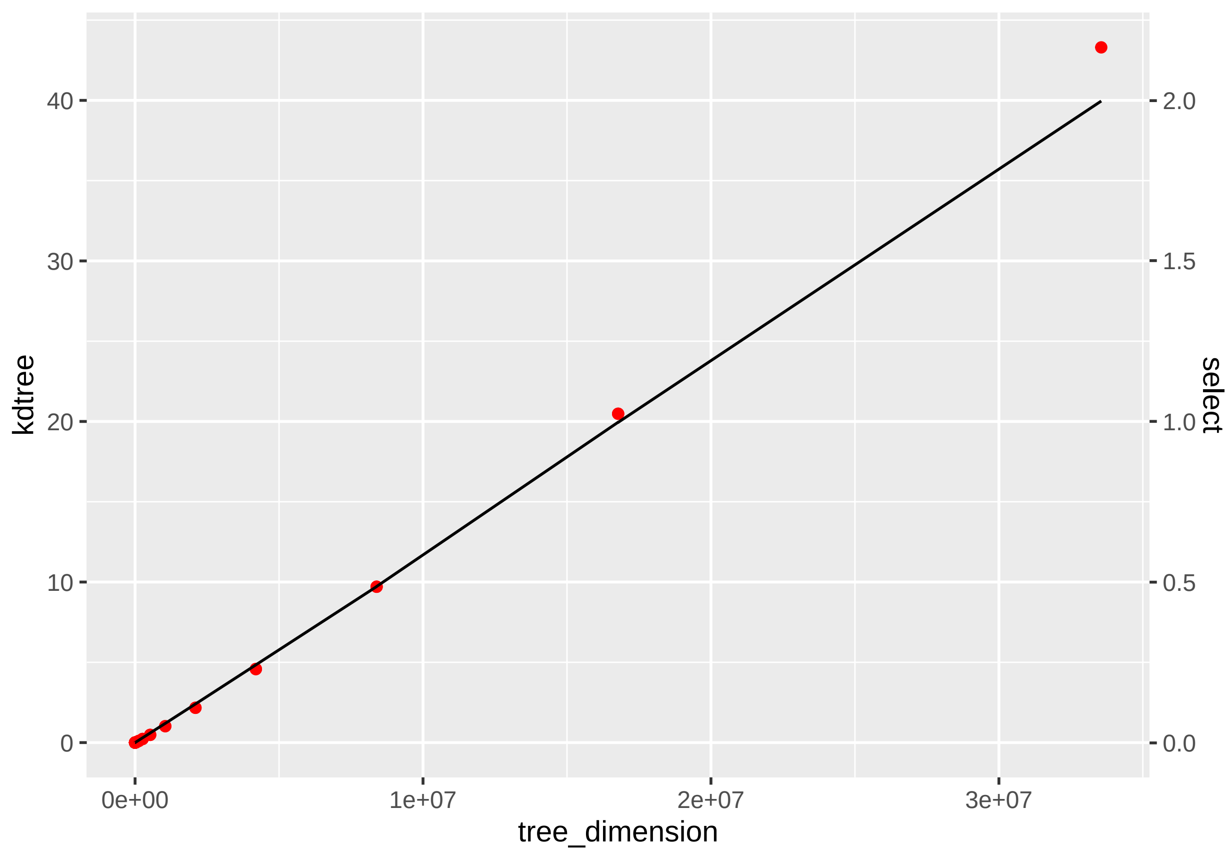
Lorenzo Trubian [mat. SM3500519]

27/02/2022

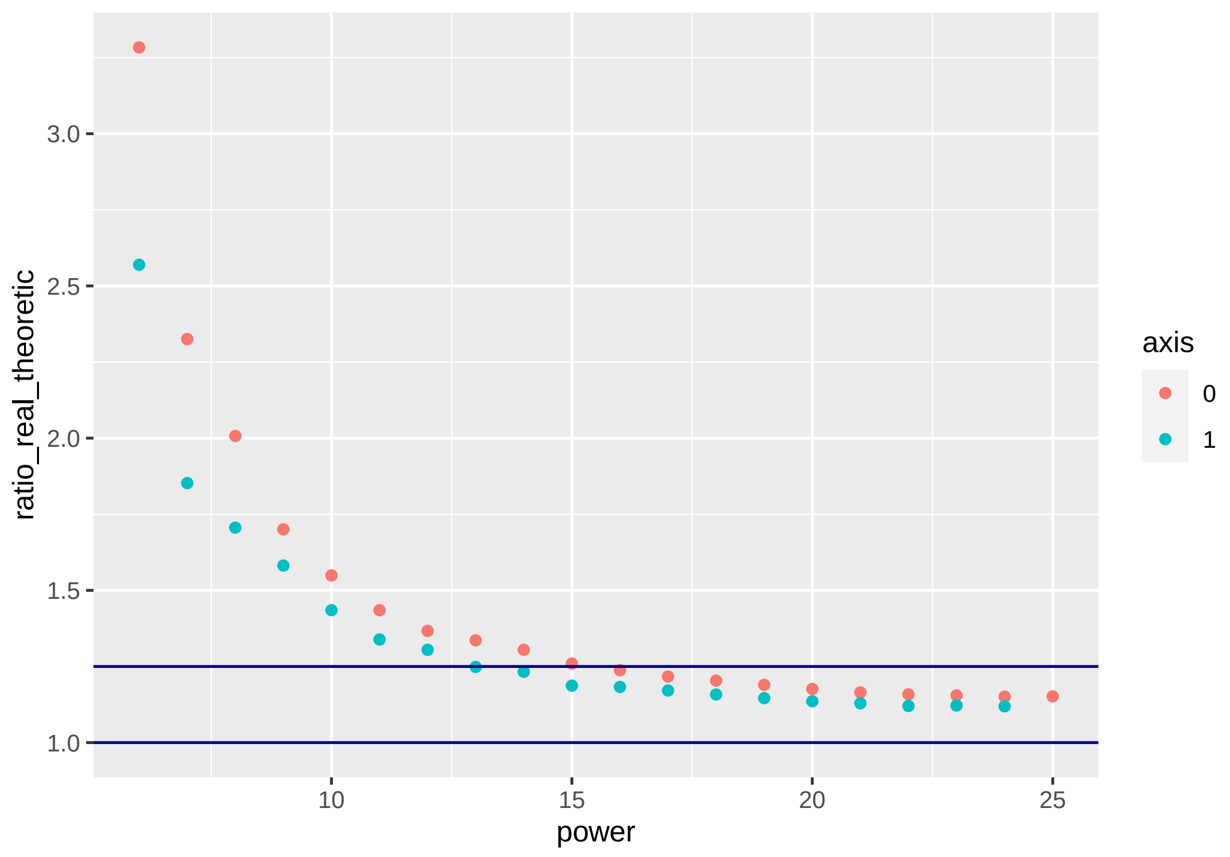
```r
setwd("~/Pubblici/learn-git/assignment2/data")
library(ggplot2)
library(gridExtra)
library(ggpubr)

tab.2 <- read.csv("parallel-mpi/time_mpi-2-0.csv")
tab.2.prec <- read.csv("parallel-mpi/time_mpi-2-1.csv")
tab.3 <- read.csv("parallel-mpi/time_mpi-3-0.csv")
tab.3.prec <- read.csv("parallel-mpi/time_mpi-3-1.csv")
tab.4 <- read.csv("parallel-mpi/time_mpi-4-0.csv")
tab.4.prec <- read.csv("parallel-mpi/time_mpi-4-1.csv")
tab.5 <- read.csv("parallel-mpi/time_mpi-5-0.csv")
tab.5.prec <- read.csv("parallel-mpi/time_mpi-5-1.csv")

tab.2$prec <- "single"
tab.3$prec <- "single"
tab.4$prec <- "single"
tab.5$prec <- "single"

tab.2.prec$prec <- "double"
tab.3.prec$prec <- "double"
tab.4.prec$prec <- "double"
tab.5.prec$prec <- "double"

total <- rbind(tab.2, tab.2.prec, tab.3, tab.3.prec, tab.4, tab.4.prec, tab.5, tab.5.prec)
total <- total[total$NUMPOINTS <= 1073741823, ]

tab.serial.prec <- read.csv("serial/time-serial_2.csv")
tab.serial <- read.csv("serial/time-serial_2-0.csv")

# tab.serial.prec
tab.serial$axis <- as.factor(tab.serial$axis)
tab.2$numproc <- as.factor(tab.2$numproc)


ggplot( data = tab.serial[,] )+
  geom_point( mapping = aes(x = tree_dimension, y = time_select)) #+
```



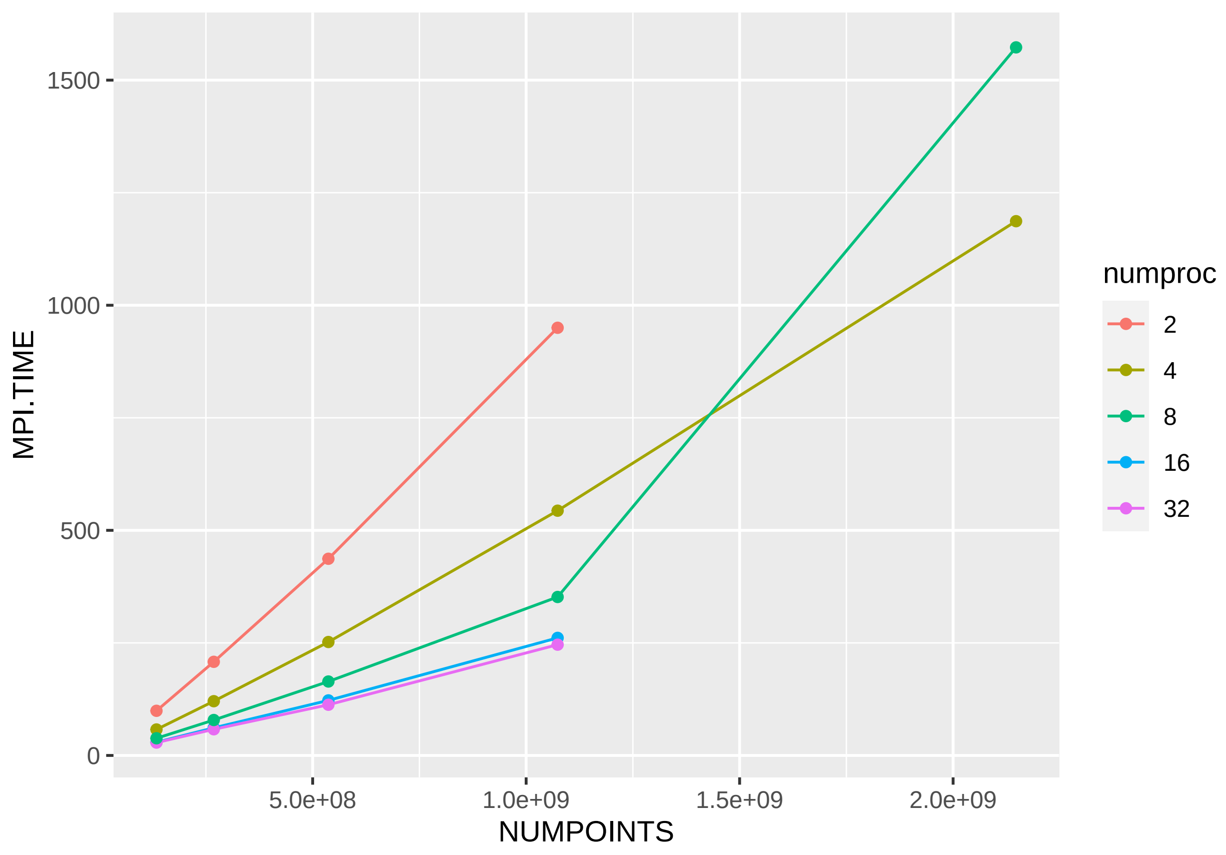
```
  # scale_x_continuous(trans='log2')

ggplot( data = tab.serial[,] )+
  geom_point( mapping = aes(x = tree_dimension, y = time_kdtree), color = "red")+
  scale_y_continuous(name = "kdtree",
                     sec.axis = sec_axis( trans=~./20, name="select"))+
  geom_line( mapping = aes(x = tree_dimension, y = time_select*20))#+
```



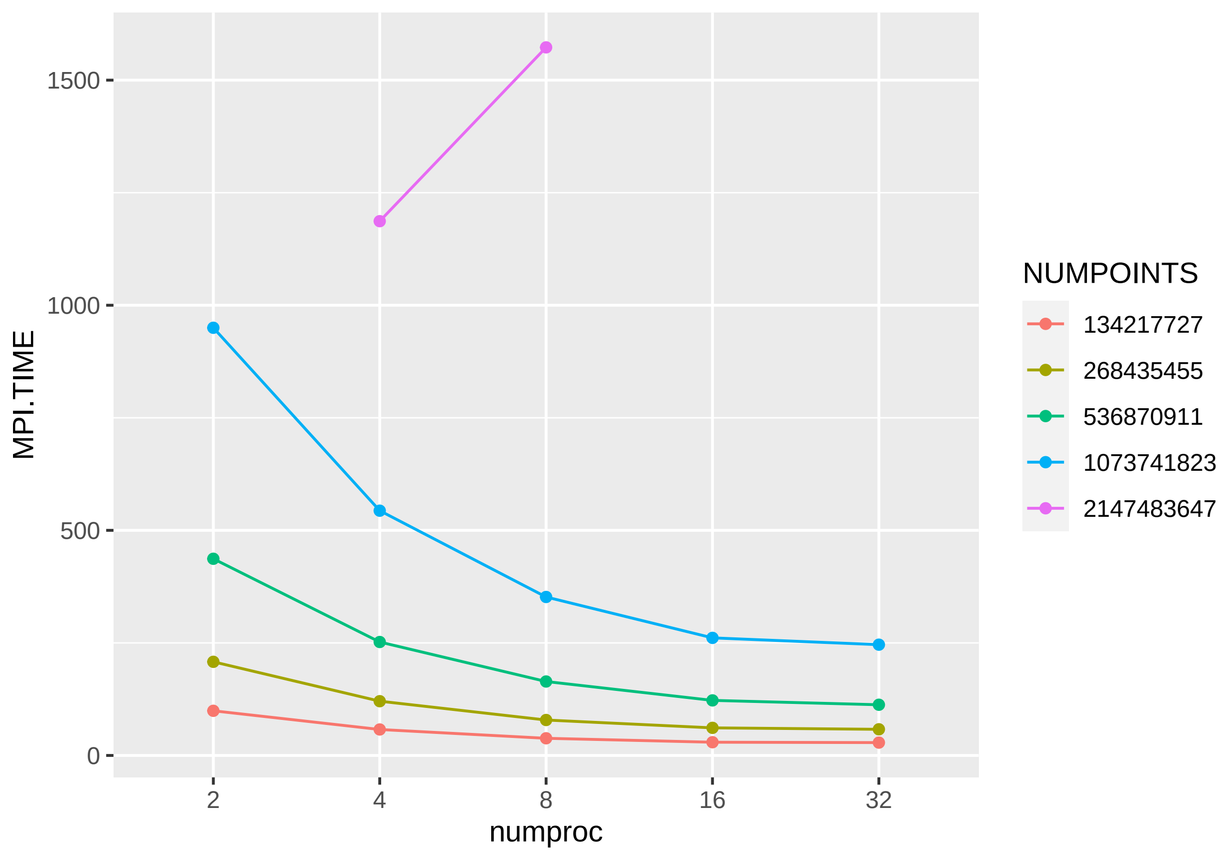
```r
  # scale_x_continuous(trans='log2')

ggplot( data = tab.serial[tab.serial$power>5,] )+
  geom_point( mapping = aes(x = power, y = ratio_real_theoretic, color = axis)) +
  geom_hline(yintercept = 1, color = "darkblue") +
  geom_hline(yintercept = 1.25 , color = "darkblue")
```



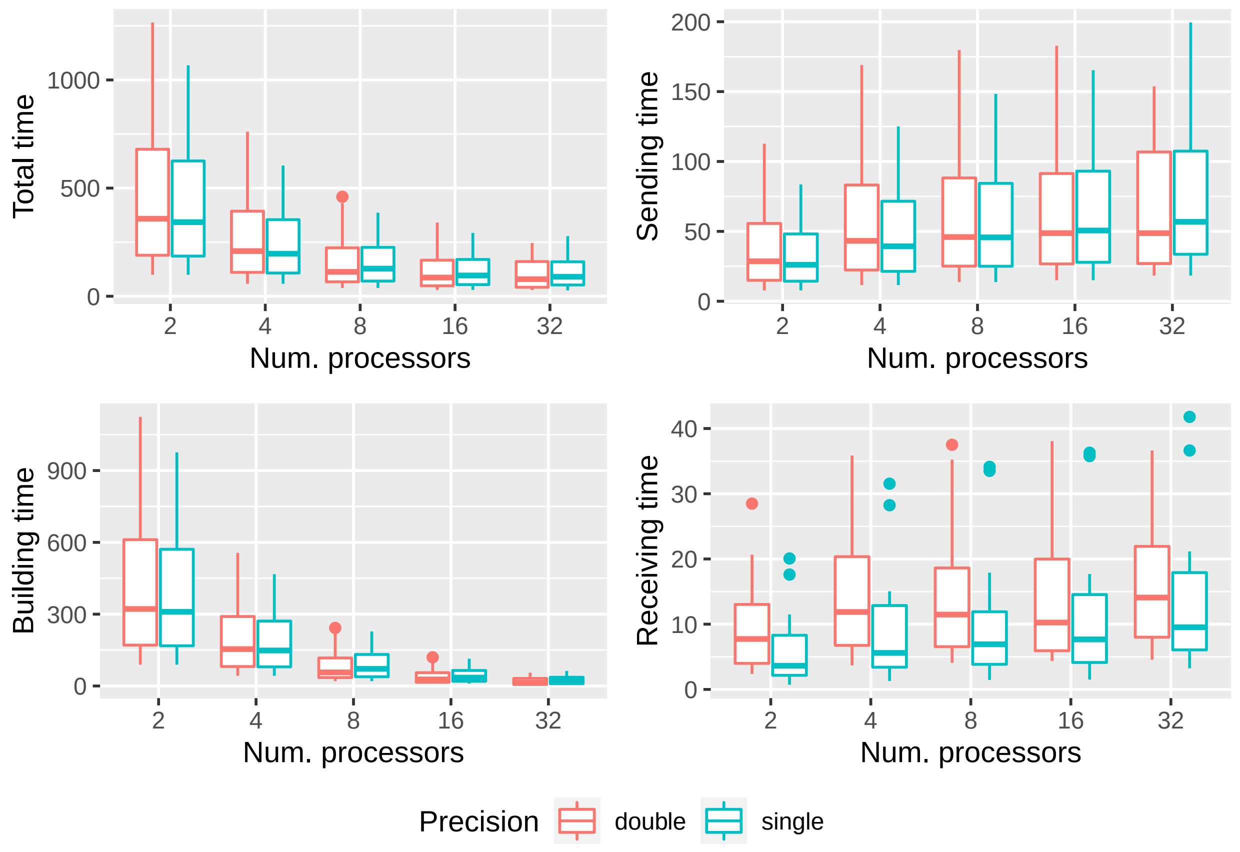
```
ggplot( data = tab.2[,] ,  mapping = aes(x = NUMPOINTS, y = MPI.TIME, color = numproc, group = numproc)
  geom_line() +
  geom_point() #+
```



```
  # scale_x_continuous(trans='log2')
# 134217727 268435455 536870911 1073741823 2147483647 4294967295

tab.2$NUMPOINTS <- as.factor(tab.2$NUMPOINTS)
ggplot( data = tab.2[,] ,  mapping = aes(x = numproc, y = MPI.TIME, color = NUMPOINTS, group = NUMPOINTS
  geom_line() +
  geom_point()
```



```
total.time <- ggplot( data = total,
        mapping = aes(x = as.factor(numproc), y = MPI.TIME, color = prec ) )+
  geom_boxplot()+
  labs( x = "Num. processors", color = "Precision", y = "Total time" )

sending.time <-ggplot( data = total,
        mapping = aes(x = as.factor(numproc), y = sending_time, color = prec ) )+
  geom_boxplot()+
  labs( x = "Num. processors", color = "Precision", y = "Sending time" )

building.time <-ggplot( data = total,
        mapping = aes(x = as.factor(numproc), y = building_time, color = prec ) )+
  geom_boxplot()+
  labs( x = "Num. processors", color = "Precision", y = "Building time" )

receiving.time <- ggplot( data = total,
        mapping = aes(x = as.factor(numproc), y = receiving_time, color = prec ) )+
  geom_boxplot() +
  labs( x = "Num. processors", color = "Precision", y = "Receiving time" )

ggarrange( total.time,
              sending.time,
              building.time,
              receiving.time,
              nrow = 2, ncol = 2 ,
              common.legend = TRUE, legend="bottom")
```



Introduction

The aim of this work was to analyze the performance improvement of building-kdtree algorithm through a parallelization. In particular, we try to investigate the difference using the distributed or shared memory approach, using MPI library in the first case and OpenMP interface in the second. Before continuing, we recall briefly the type of structure that we want to obtain. Given a d-dimensional set of points, a kdtree is a binary and balanced tree where each node contains a point which is the median (according one chosen dimension) of all the points in the sub-tree having the node as the root.

Algorithm

In order to construct the kdtree, an algorithm is needed to find the median of a list of values. In this work it is used the *median of medians* algorithm as it is described here or in the *Introduction to Algorithms* [1] to find the median of a given set of kpoint along a fixed axis. The worst-case running time of this algorithm, which will be called **select**, is linear.

We can describe the building-kdtree algorithm with the following recursive procedure:

1. given a list of kpoint and the split axis, find the median;

[1]Section 9.3 of *Introduction to Algorithms* Third Edition, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.

2. make a partition of the list around the median;
3. save the median and update the split axis;
4. call recursively the procedure on the two sub-lists.

The **select** algorithm actually solves both the first and the second point. The described sequence of steps allow a clear parallelization: whenever a compute unit reaches the 4th step, the recursive call could easily be assigned to another compute unit. This is what is implemented, with some slight differences, both in the shared and distributed memory approach.

Implementation

Both with MPI library and OpenMP interface, the algorithm passes through two main phases:

- during the first phase the code continues to assign the two sub-list created with the 4th step to different compute unit;
- during the second phase, however, the code become completely serial and the jobs on the two sub-lists are completed by the same compute unit that creates them.

MPI, distributed memory