Foundations of High Performance Computing

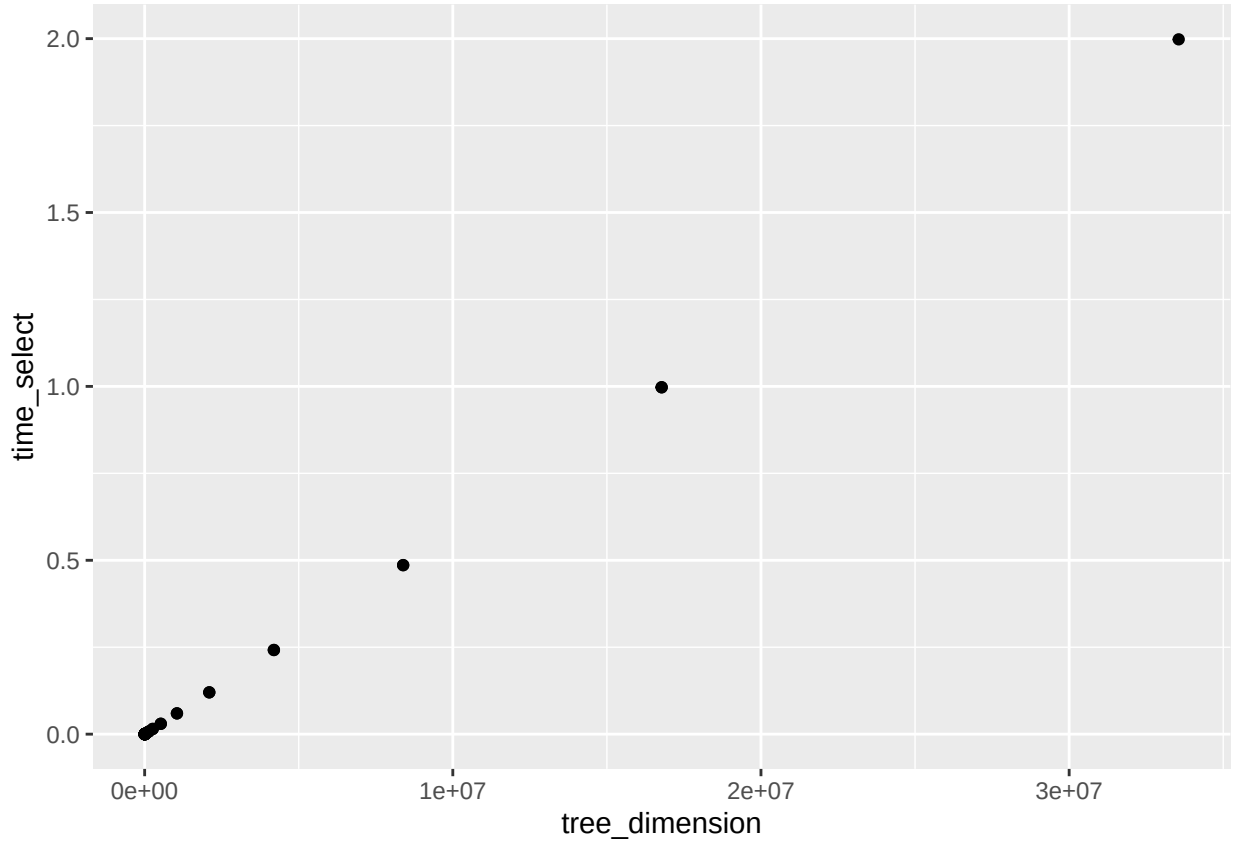
Assignment II

Lorenzo Trubian [mat. SM3500519]

27/02/2022

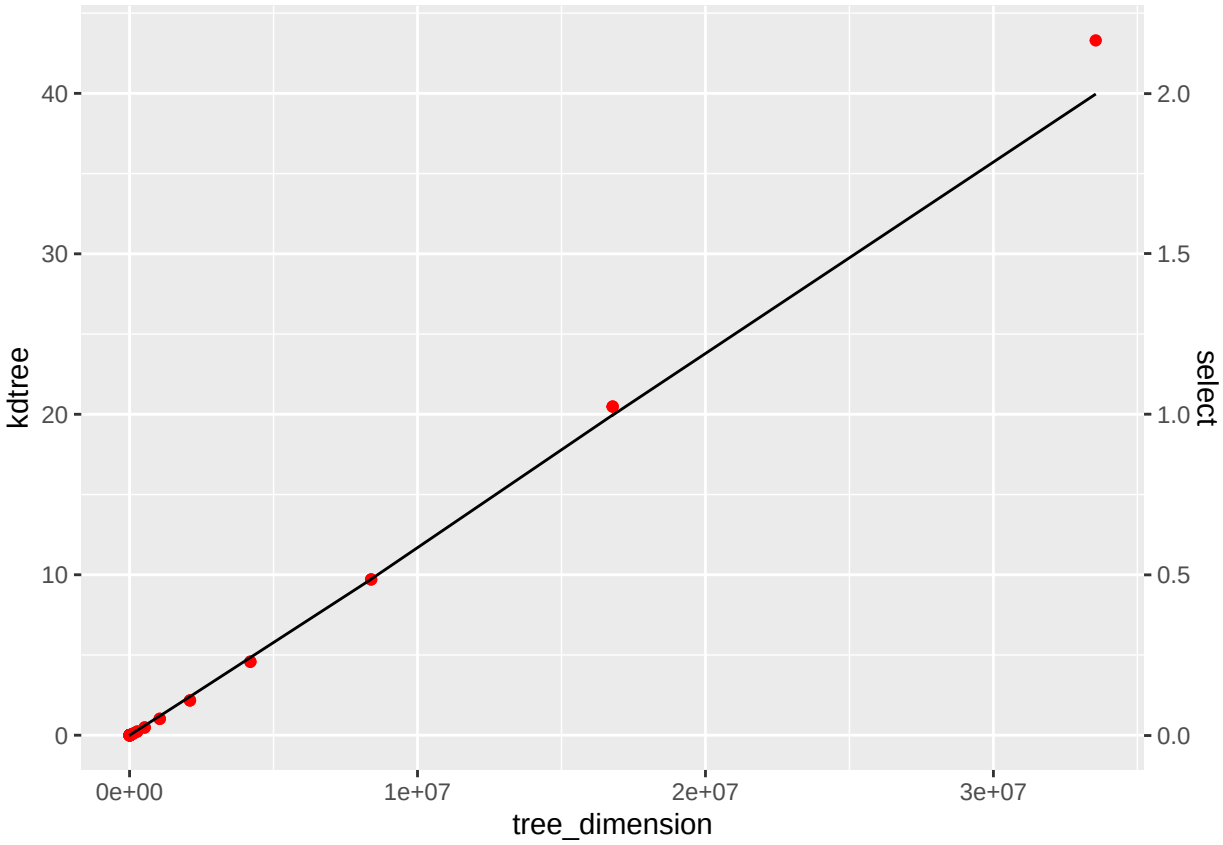
```
setwd("~/Pubblici/learn-git/assignment2/data")
library(ggplot2)
tab.2 <- read.csv("parallel-mpi/time_mpi-2-0.csv")
tab.2.prec <- read.csv("parallel-mpi/time_mpi-2-1.csv")
tab.serial <- read.csv("serial/time-serial_2.csv")
# tab.serial.prec
tab.serial$axis <- as.factor(tab.serial$axis)
tab.2$numproc <- as.factor(tab.2$numproc)


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



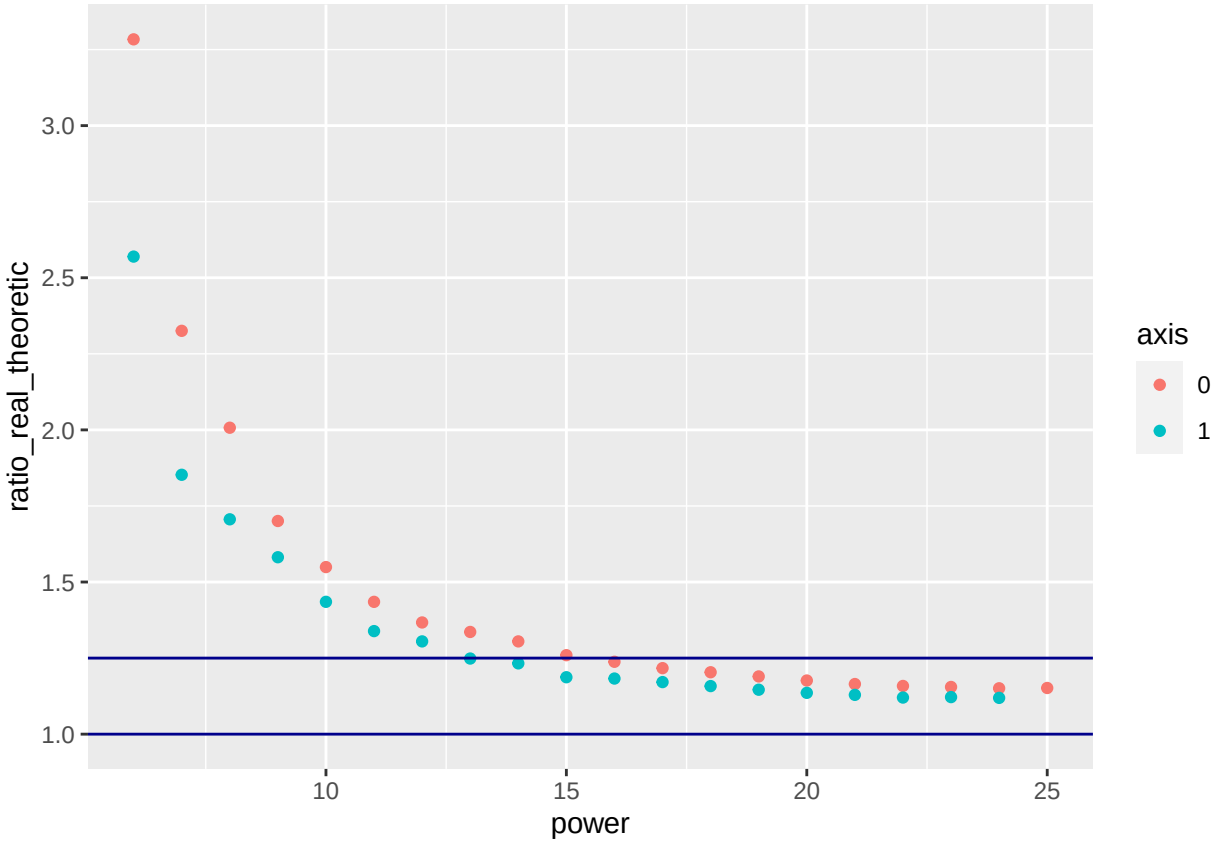
```
  # scale_x_continuous(trans='log2')

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))#+
```

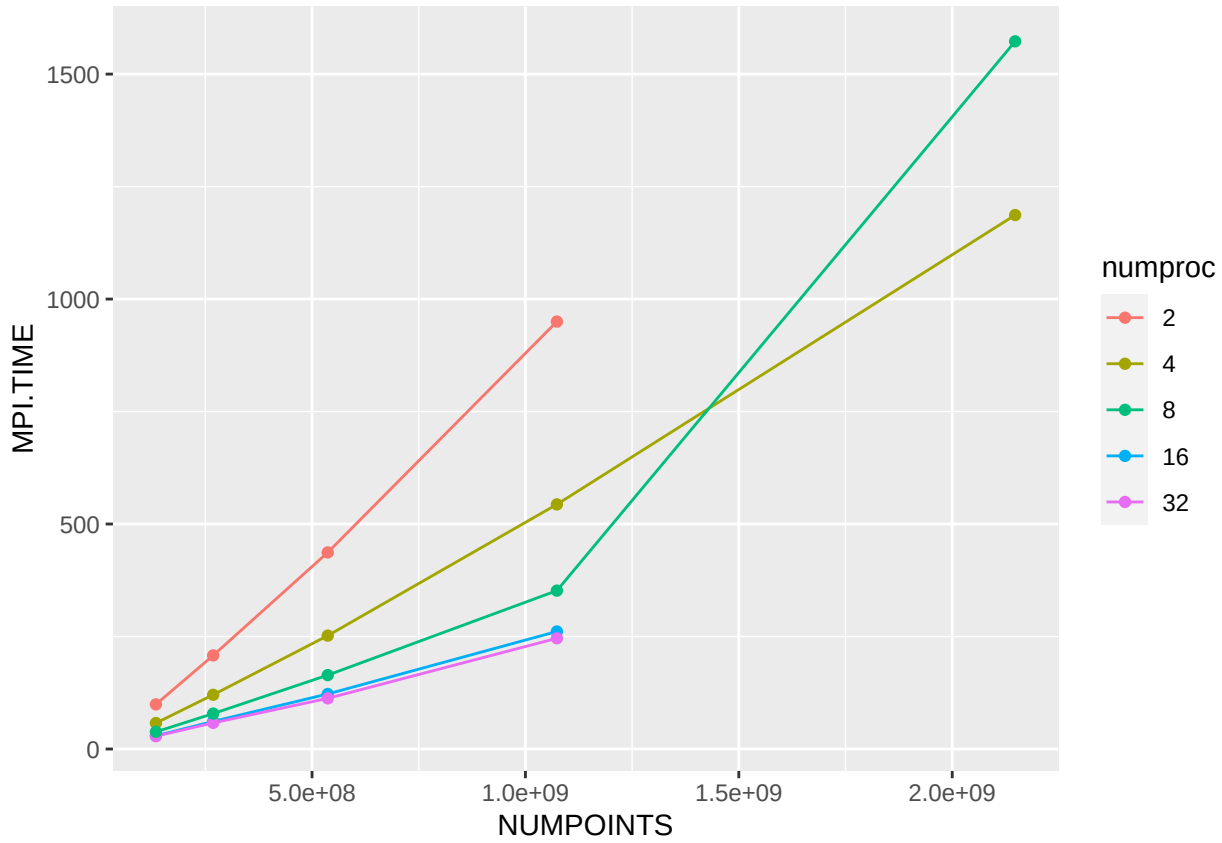



```
  # scale_x_continuous(trans='log2')

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")
```

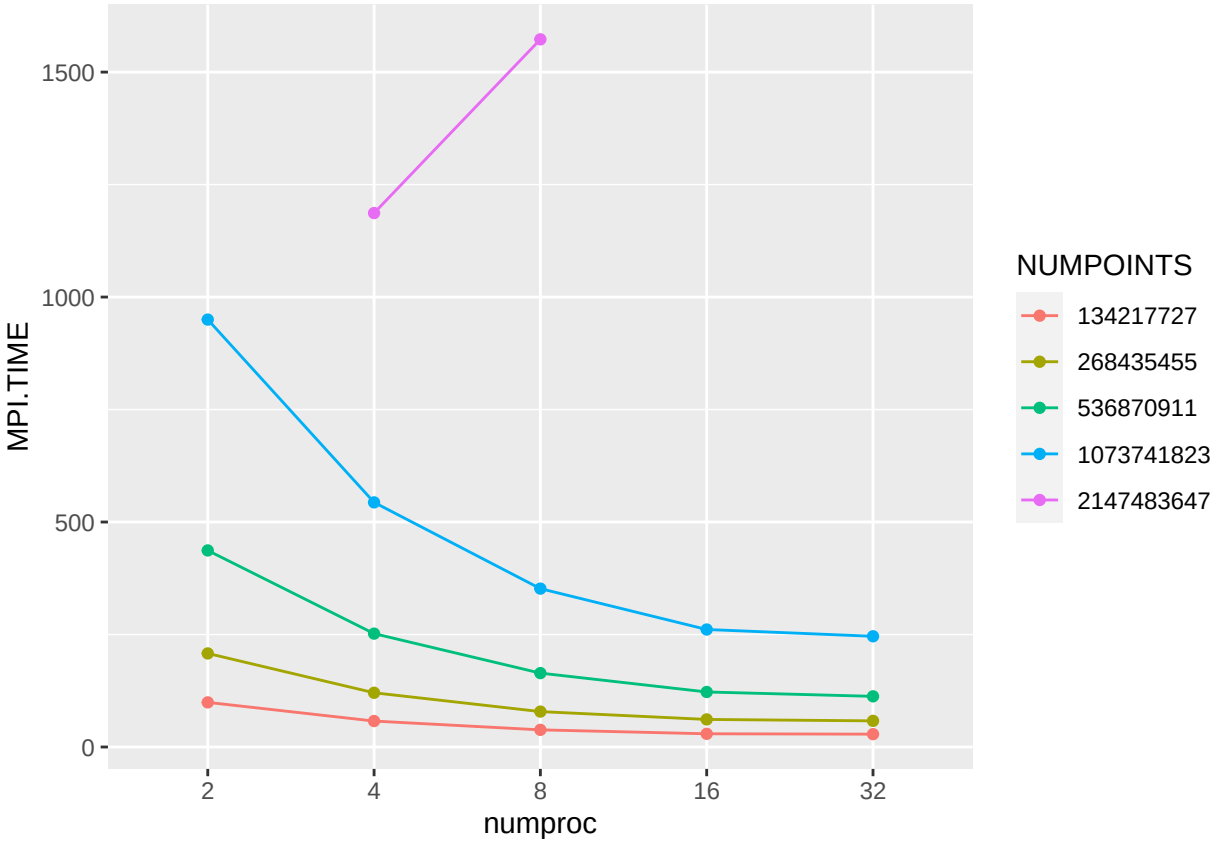



```
ggplot( data = tab.2[,] ,  mapping = aes(x = NUMPOINTS, y = MPI.TIME, color = numproc, group = numproc))
  geom_line() +
  geom_point() #+
```




```
  # 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()
```



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 API 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, it is necessary an algorithm to find the median of a list of value. 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;
2. make a partition of the list around the median;
3. save the median and update the split axis;

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

4. call recursively the procedure on the two sub-list.

Then

Implementation