POLITECNICO
MILANO 1863

OpenMP DEBS 2013

https://github.com/marckw94/OpenMP-DEBS-2013

Middleware Technologies for Distributed Systems
Prof. Mottola Luca

Paola Sanfilippo 882892
Francesco Tinarelli 883738
Marco Wenzel 883732

Index

Introduction

• Goal

The goal of the project is to display the ball possession for each player and for each team in a soccer match, using OpenMp and/or MPI.

• Definitions

The program takes as input 2 parameters, T and K:
•T is a variable that expresses the amount of seconds that are between each update of the ball possession.
•K is the maximum distance for which we can say that the ball possession belongs to a certain player.

Assumptions

We decided to work on data aggregated by player and ball, and to do so, we preprocess the “full-game.csv” file.
To reach our goal we imagined a hypothetical real time program in which the data arrives every T seconds (see Definitions paragraph) divided according to the player. We then disaggregated the full-game.csv file, creating (if it haven’t been already created) a sensors folder, with the data divided for each player (player’s name.csv) and for the ball (ball.csv), filtered also excluding all the moments in which there’s no game (before the match, first and second half, end of the game and game interruptions). These files are composed of the respective timestamps and coordinates, generated by the sensors. All the timestamps are rewritten from picoseconds to a millisecond accuracy.
For the ball possession, we know that the data arrives in chronological order: whenever we’re looking for the position of a player, we’re going to take the position related to the closest past timestamp.

Technologies

We used CLion as development environment and MinGW with OpenMP and Boost libraries for file managing.
We chose to avoid the usage of MPI because we have 17 files with quite little dimensions: working on a distributed system, MPI would add an overhead to every communication between processes, invalidating the performances.
Using OpenMP, we parallelized on threads the computation. We used a Dell Inspiron 7559 with a Intel core i7 quad-core processor, that allowed us to use up to 8 threads.

Application design

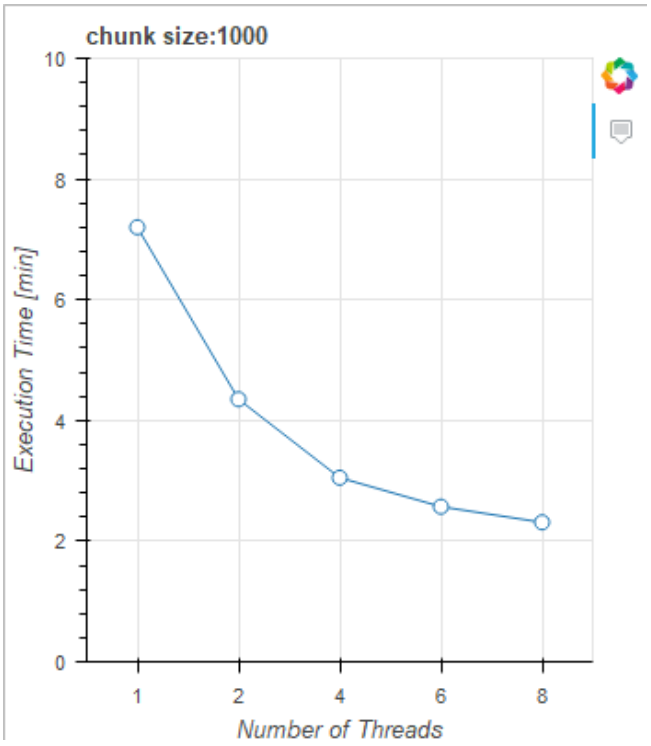
• Architecture

Initially there's a phase in which we preprocess the data, if needed. Through the Datas.cpp class we load all the metadata, found in the metadata file, of the players, the ball, the referee and the game's interruptions.
Once everything has been loaded, the data is disaggregated in the players and ball files (see full-game class).
Last we process the data using the run method in the main class.

• Algorithms

Let's focus on the processing of data.
First, through the Ball class, we load a chunk of 1 second of data of the game. The same operation is performed for every player: the Player.cpp class loads a chunk of data of the same second of the game. The loading of the ball chunk takes place in a single thread, while the loading of the players' chunks is parallelized with a thread for each player.
Once the chunks have been loaded, we start with the proper analysis of the ball possession, that is performed with a loop on the ball's chunk with parallel iterations.
The loop works like this: for each timestamp of the ball, we take the related position and we compare it with the position of each player, that has a timestamp equal or preceding the ball's one. We then assign the ball possession to the player that has the lowest distance (i.e. the difference between the 2 positions), only if it is less than the chosen K. The assignment of the ball possession is made by increasing the player's tick, that is a millisecond of possession. The increase of the tick is synchronized among all the threads, in order to avoid the race condition.
Every time that T chunks of data are analyzed, the result is print.

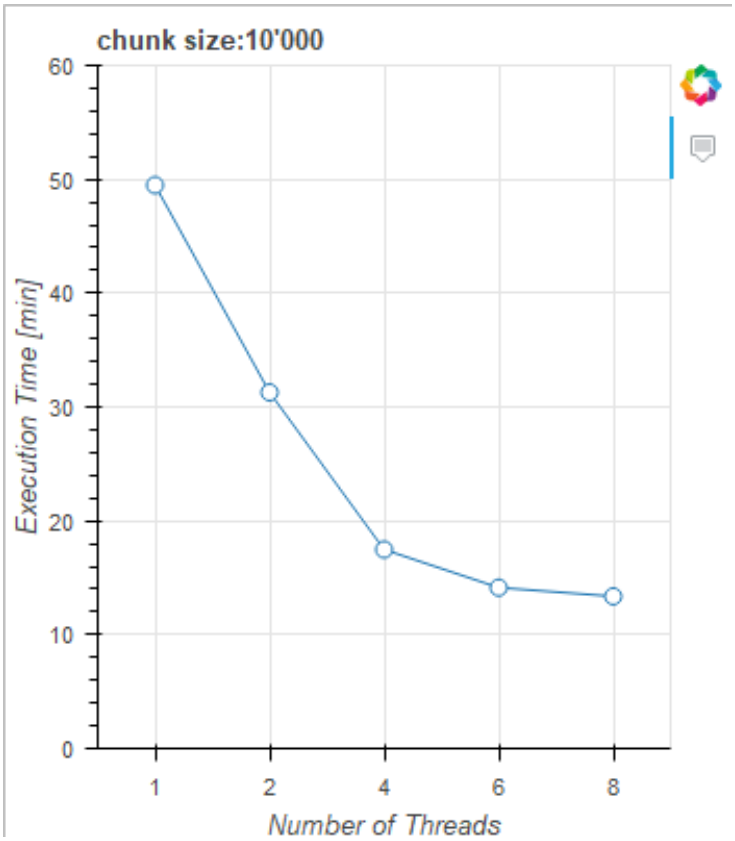
Analysis of the results

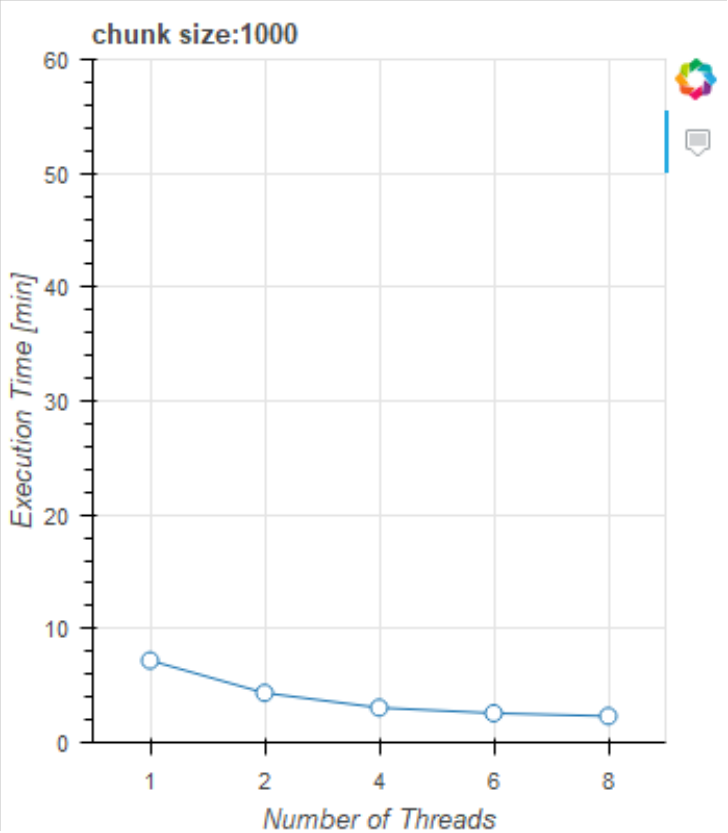


These are the results for T = 60 seconds and K = 2 meters, using different amounts of threads.
As we can see from the graph, there's a speedup when we increase the number of threads. The speedup is meaningful up to 4 threads, instead the improvement decreases for more than 4 threads. Increasing even more the number wouldn't be effective, so we suggest to use 8 threads.

Conclusions

We performed several tests on the same program to verify how the parallelization influences the performances, varying the amount of thread up to make the program totally sequential.
As observed in the analysis of the results, the usage of the threads improves the performance of almost 25% of the time until it stabilize to a number of threads too high.
Another observation done during the software development is that the performances depend on the chosen size chunk, because of the data locality of of the array in memory.
This is the graph for T = 60 seconds and K = 2 meters but with a chunk size equals to 10000 seconds





A final consideration can be done on the difference of the frequency of the ball and the players.
In fact since the ball has a frequency 10 times bigger than the one of the players, if we skip 50% of the data, we obtain a speedup of the 50% while achieving almost the same results that we would have using all the data.

```
End game results
Possession time of Nick Gertje 01:32.660
Possession time of Dennis Dotterweich 01:31.995
Possession time of Niklas Waelzlein 01:40.381
Possession time of Wili Sommer 03:22.824
Possession time of Philipp Harlass 01:54.426
Possession time of Roman Hartleb 01:16.035
Possession time of Erik Engelhardt 02:04.186
Possession time of Sandro Schneider 01:54.919
Possession time of Kevin Baer 01:58.297
Possession time of Luca Ziegler 01:35.587
Possession time of Ben Mueller 01:32.366
Possession time of Vale Reitstetter 02:48.591
Possession time of Christopher Lee 01:34.140
Possession time of Leon Heinze 00:52.678
Possession time of Leo Langhans 01:57.284
Possession time of Leon Krapf 01:35.765
Total ticks team A = 13:22.507
Total ticks team B = 15:49.627
NumberOfPrint = 69
Time of analysis = 02:31.327 number of thread = 8
```

```
End game results
Possession time of Nick Gertje 01:32.570
Possession time of Dennis Dotterweich 01:32.0044
Possession time of Niklas Waelzlein 01:40.390
Possession time of Wili Sommer 03:23.054
Possession time of Philipp Harlass 01:54.544
Possession time of Roman Hartleb 01:16.142
Possession time of Erik Engelhardt 02:04.224
Possession time of Sandro Schneider 01:54.928
Possession time of Kevin Baer 01:58.142
Possession time of Luca Ziegler 01:35.810
Possession time of Ben Mueller 01:31.926
Possession time of Vale Reitstetter 02:48.238
Possession time of Christopher Lee 01:33.888
Possession time of Leon Heinze 00:52.684
Possession time of Leo Langhans 01:56.978
Possession time of Leon Krapf 01:35.960
Total ticks team A = 13:22.928
Total ticks team B = 15:48.554
NumberOfPrint = 69
Time of analysis = 01:50.445 number of thread = 8
```