Algorithm Analysis

nabil A.

December 4, 2015

Introduction

For the author's portion of the Mini-Project, he wrote an implementation of four algorithms:

- Bellman-Ford
- Dijkstra's
- and an OpenMP version of each

Given a fully-connected graph, these four algorithms were then used to calculate shortest-paths from Source Node 0 to one hundred other nodes in the network, and parallized with 2, 3, and 4 threads.

In the following, we seek to explore the results of this timing comparison, and elaborate on some of the findings.

Analysis

Data Description

As outlined in the shell script `commands.sh`, all the timing results were stored in the file `results.txt`, which we can see the contents of:

```r
suppressPackageStartupMessages({
  library(readr)
  library(tidyr)
  library(dplyr)
  library(ggplot2)
})
```

```
## Warning: package 'readr' was built under R version 3.1.3
```

```r
# read in the file and inspect it
aa <- read_delim("~/monitorMP/monitorSearch/monitorSearch/results.txt", delim = "\t")
aa %>% glimpse
```

```
## Observations: 600
## Variables: 8
## $ Type   (chr) "Dist", "Time", "Dist", "Time", "Dist", "Time", "Dist",...
## $ BF     (dbl) 2514.61400, 0.00822, 2514.61400, 0.00812, 2514.61400, 0...
## $ DK     (dbl) 2514.61400, 0.00804, 2514.61400, 0.00793, 2514.61400, 0...
## $ OMPBF  (dbl) 2514.61400, 0.00902, 2514.61400, 0.00903, 2514.61400, 0...
## $ OMPDK  (dbl) 2514.61400, 0.00911, 2514.61400, 0.00929, 2514.61400, 0...
## $ FROM   (int) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ TO     (int) 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 4...
## $ NTHRDS (int) 2, 2, 3, 3, 4, 4, 2, 2, 3, 3, 4, 4, 2, 2, 3, 3, 4, 4, 2...
```

Note that there are 8 columns:

- `Type` indicates the type of value recorded for that row (distance between the two nodes, or execution time)
- The next four columns contain values for the four algorithms
- `FROM` and `TO` indicate the source and destination nodes the algorithms were run with
- `NTHRDS` indicates the number of threads with which the OpenMP versions were run.

This data can be tidied to facilitate subsequent analysis:

```
algo_times <- aa %>%
  gather(Algo, Value, BF:OMPDK) %>% select(Algo, everything())
algo_times
```

```
## Source: local data frame [2,400 x 6]
##
##      Algo  Type  FROM    TO NTHRDS      Value
##    (fctr) (chr) (int) (int)  (int)      (dbl)
## 1      BF  Dist     0     1      2 2514.61400
## 2      BF  Time     0     1      2    0.00822
## 3      BF  Dist     0     1      3 2514.61400
## 4      BF  Time     0     1      3    0.00812
## 5      BF  Dist     0     1      4 2514.61400
## 6      BF  Time     0     1      4    0.00809
## 7      BF  Dist     0     2      2 2392.98300
## 8      BF  Time     0     2      2    0.00831
## 9      BF  Dist     0     2      3 2392.98300
## 10     BF  Time     0     2      3    0.00808
## ..    ...   ...   ...   ...    ...        ...
```

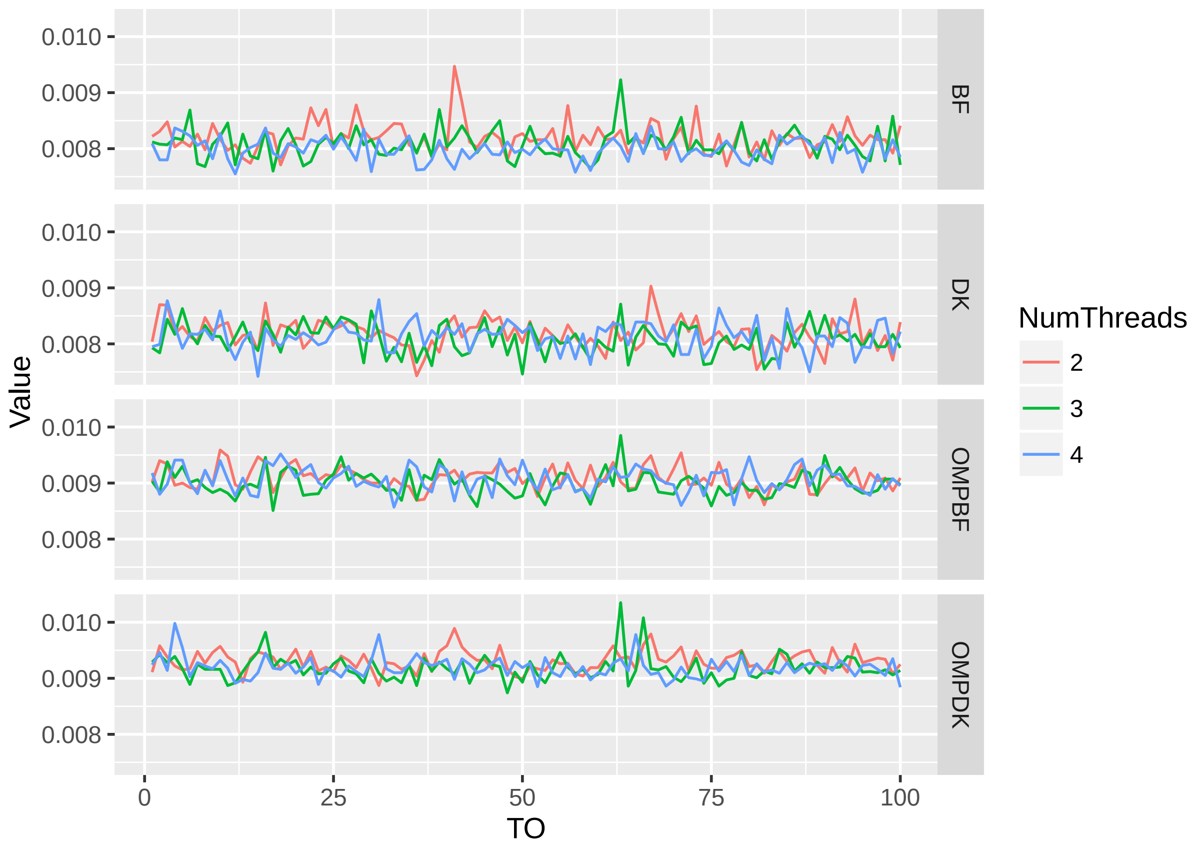
from which we see that the same information is still there, just differently arranged.

Visualization

Because there is so much that can be investigated with the data, we first start with a broad question: How does the runtime vary by algorithm and number of threads used?

Consider the following graph:

```
algo_times %>%
  filter(Type == "Time") %>%
  mutate(NumThreads = factor(NTHRDS)) %>%
  select(-NTHRDS) %>%
  ggplot(aes(TO, Value, color = NumThreads)) + geom_line() +
  facet_grid(Algo ~ .)
```



Here, the y-axis represents the time needed to run the algorithm, the x-axis is the destination node used (Recall that the source node in all of these runs was Node 0), and the four panels respectively show results for Bellman-Ford, Dijkstra's, OMP Bellman-Ford, and OMP Dijkstra's. Note that due to the arrangement, it's relatively easy to compare times for all thread-algorithm combinations; just consider a vertical line (corresponding to a single `TO` value) through all four panels.

From the above graph, a few general observations can be made:

1. The OMP versions seem to fare almost exclusively worse than the serial versions
2. Among the OMP algorithms, using more threads doesn't seem to significantly improve times much
3. For this (fully-connected) graph, Bellman-Ford's performs about as well as Dijkstra's.

Concerning the first point, it's not clear to what extent the lack of real improvement is due to a poor implementation of parallelization on the author's part.

As for the second point, although there is (naturally) variation in the performance of the parallel algorithms when different thread counts are used, this variation doesn't appear to be much more than random noise. Indeed, for these generated results, there is no number of threads which performs uniformly better than any other number. The author thinks that one possible reason for this could be the nature of the algorithms themselves. Recall that basically the only parallelization occurs in the relaxation stage of Bellman-Ford's and Dijkstra's, and those constitute critical sections of the code. Hence, it is precisely those parts which are parallelized which benefit only marginally from the parallelization (whereas the rest isn't affected at all). Thus, it makes sense that further increasing the number of threads used adds little value (if any) to the performance of the algorithms.

Concerning the third point, the author believes that the similarity in performance is likely due to how connected the graph is. That is, even though Dijkstra's is typically much faster, when the graph is fully connected, then at every iteration, each remaining has parameters updated (i.e., each node is checked). This behavior is similar to Bellman-Ford's, wherein the nodes are checked at each iteration as well. Thus, we can reasonably expect that with sparser graphs, the performance of Dijkstra's algorithm would outperform Bellman-Ford's to a much greater extent.

Conclusion

In the aforementioned, we were able to explore and compare some performance results of two algorithms and their parallelized counterparts. However, due to the very serial nature of BF's and Dijkstra's, parallelization failed to yield significantly better results. One area where the above findings could be further examined, is by comparing performance of these algorithms while varying the connectivity of graphs used in the computations.