# Parallel and Distributed Systems
# Vertexwise triangle counting

Θεόδωρος Κατζάλης
AEM:9282
katzalis@auth.gr

6/12/2020

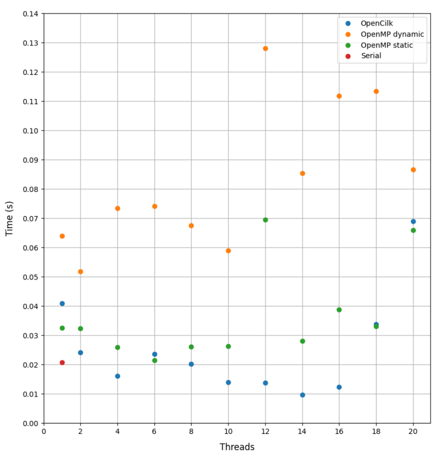
## Contents



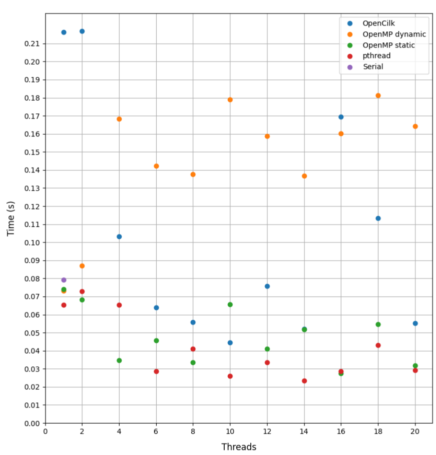
## 1 Introduction

About **vertexwise triangle counting**, searching was a common issue for all the versions. The v3 is implemented with binary and the v4 with binary and linear algorithms, trying to find optimum results. Time executions of parallel versions are depicted along with a reference point of the equivalent serial implementation.

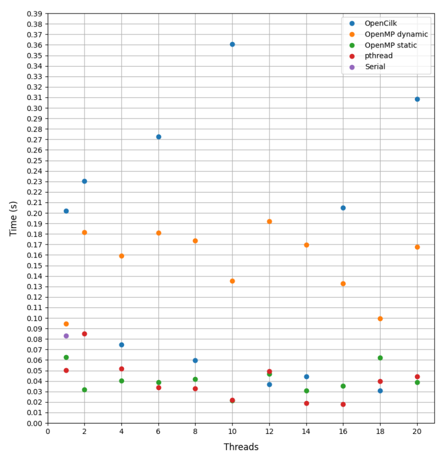
## 2 n = 1.441.295, m = 3.099.940 (belgium_osm.mtx)




(a) V3 binary search

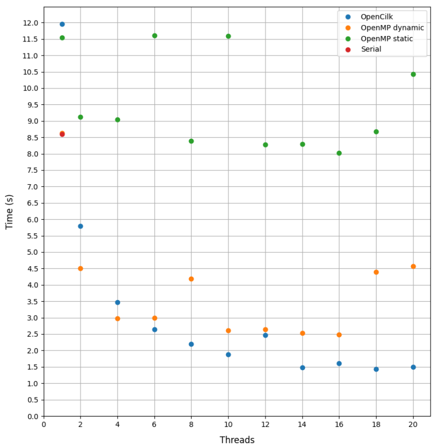



(b) V4 binary search

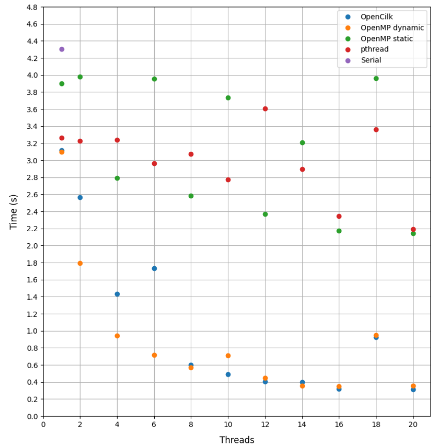



(c) V4 linear search

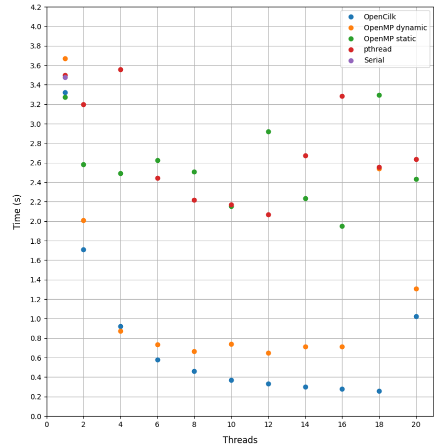
# 3 n = 1.134.890, m = 5.975.248 (com-Youtube.mtx)




(a) V3 binary search

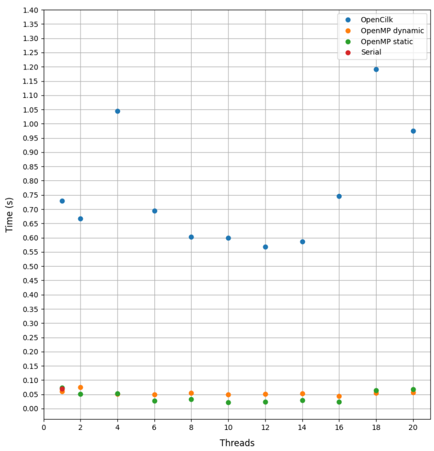



(b) V4 binary search

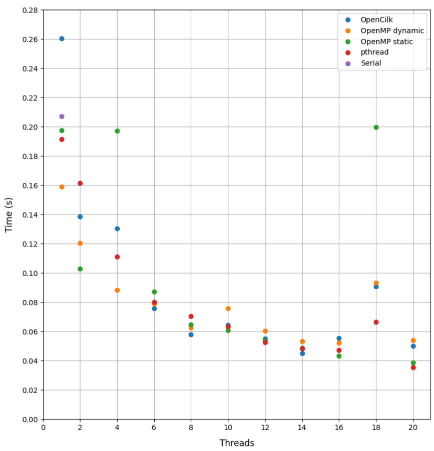



(c) V4 linear search

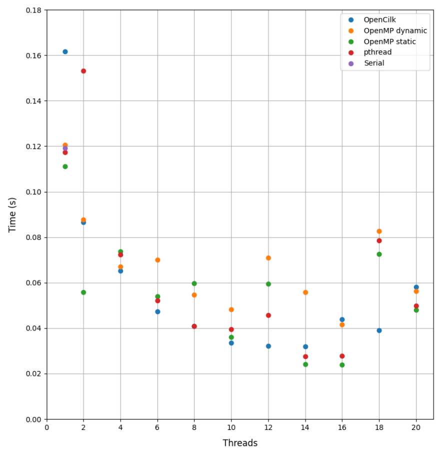
# 4 n = 326,186, m = 1,615,400 (dblp-2010.mtx)




(a) V3 binary search

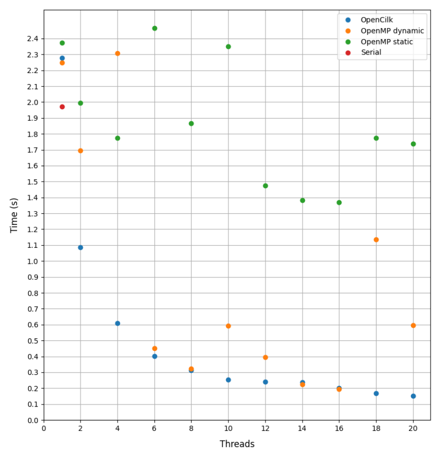



(b) V4 binary search

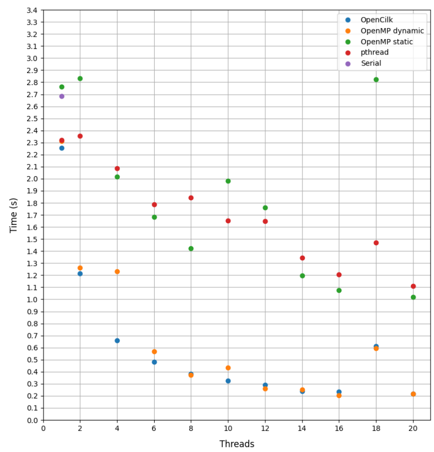



(c) V4 linear search

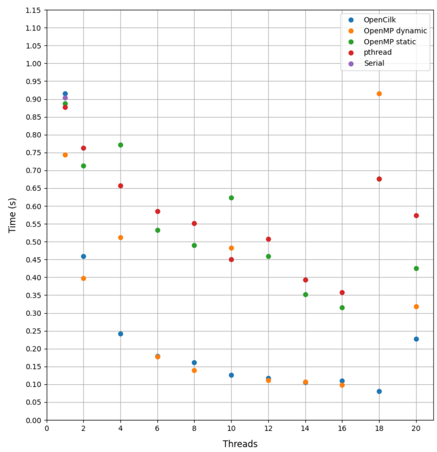
# 5 n = 6.143 , m = 1.227.742 (mycielskian.mtx)




(a) V3 binary search

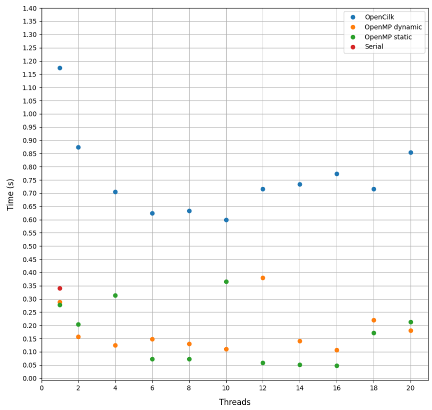



(b) V4 binary search

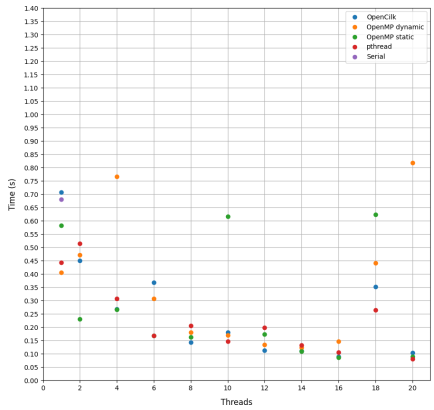



(c) V4 linear search

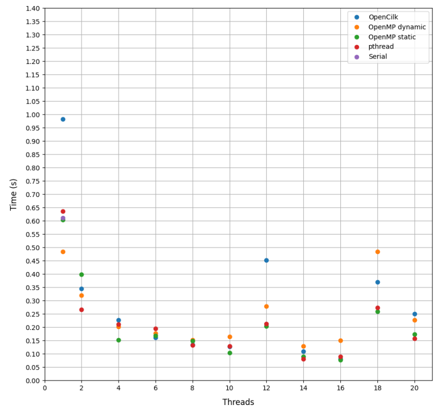
# 6 n = 1.039.183 , m = 6.229.636 (NACA0015.mtx)




(a) V3 binary search




(b) V4 binary search




(c) V4 linear search

# 7 Conclusions

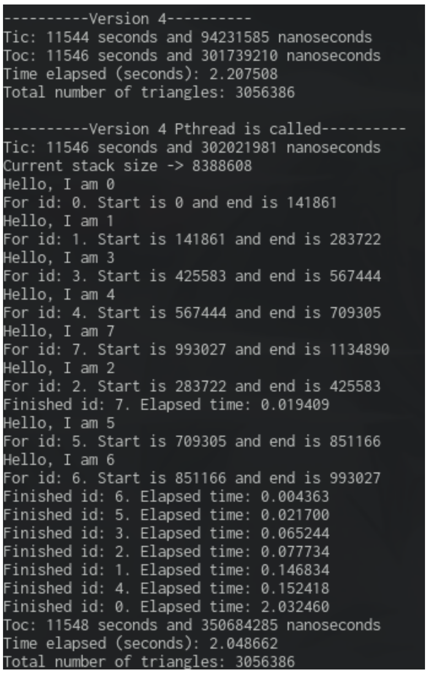
- "**Οὐκ ἐν τῷ πολλῷ τὸ εὖ**". More threads won't result always to better performance. Speedup efficiency may be better with a smaller number. It's all about finding the golden ratio between load (matrix characteristics) and thread overhead.
- Parallelism with **one thread** is most of the times worse than the sequential. This is normal due to the additional load of thread creation and setting up the run time environment.
- **Scheduling**: static vs dynamic. Indeed scheduling is a crucial performance factor. In some cases the one outperforms the other. Calculating chunk size on the fly (dynamic) is appropriate when the computational time and the load is unknown. On the other hand static scheduling is suitable for load balanced across the iterations.
- Using pthreads has similar behaviour with OpenMP static scheduling. This is reasonable because under the hood, OpenMP is actually based on pthreads and the scheduling algorithm, by default, divides equally the load to the number of the available threads. The same concept is applied to our v4 pthread implementation.
- The structure of v3 is such that the parallel versions could result to a **data race** opposed to v4. To avoid that, we needed **mutual exclusion**. So we used a list from the Cilk Reducer Library and a reduction array for the OpenMP. These two things however, as we can see in some cases (NACA0015.mtx and dblp-2010.mtx), is the root cause of **poor** performance. Another possible way, to avoid so much syncing, is to create an array of the critical variable, so each thread will have its own copy in the heap. Such an approach is vulnerable to false sharing though and isn't a portable solution.
- **Embarrassingly parallel** v4. The v4 is a very good candidate for parallelism because there is no risk of data race and there is no need of mutexes and any kind of heavy syncing. Each thread is writing to a different memory location. For these reasons v4 in most cases has better performance.
- Regarding v4, binary and linear search in sorted lists have similar performance for the given test cases.

All the results derived using AUTh's HPC infrastructure. The compiler that is used was gcc 7.5.0 to be compatible with the cilk plus run time library. The **Github repo** for the source code can be found here.

# 8 Load balancing

The more the loops are balanced, the better our **static** implementation will be. Για παράδειγμα, παρατηρώντας τις δύο παρακάτω εικόνες[1]. το πρώτο thread κάνει σχεδόν όλη την δουλειά προκαλώντας load imbalance και επηρεάζοντας σημαντικά το speedup. Στα πλαίσια του χρονικού διαστήματος της εργασίας, λείπει μια δυναμική έκδοση των pthreads. Εικάζουμε ότι σε τέτοιες περιπτώσεις θα ήταν αρκετά πιο αποδοτική (βλέποντας και την απόκριση του schedule dynamic σε openmp υλοποίηση).

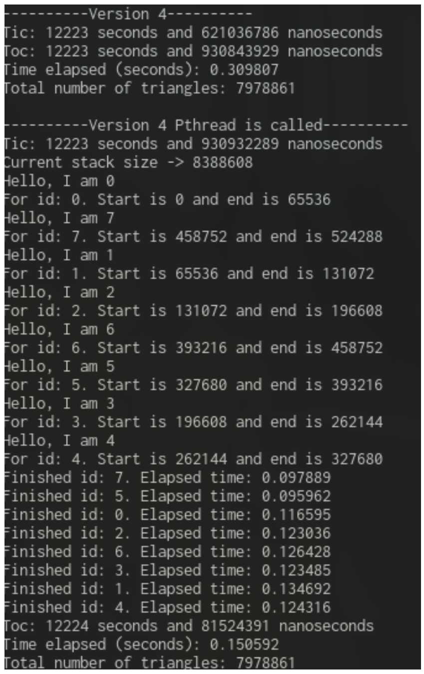
Με αυτά τα γραφήματα μπορούμε να εξηγήσουμε την διαγορετική απόδοση μεταξύ dynamic και static scheduling στους παραπάνω πίνακες.



```
----------Version 4----------
Tic: 11544 seconds and 94231585 nanoseconds
Toc: 11546 seconds and 301739210 nanoseconds
Time elapsed (seconds): 2.207508
Total number of triangles: 3056386

---------Version 4 Pthread is called---------
Tic: 11546 seconds and 302021981 nanoseconds
Current stack size -> 8388608
Hello, I am 0
For id: 0. Start is 0 and end is 141861
Hello, I am 1
For id: 1. Start is 141861 and end is 283722
Hello, I am 3
For id: 3. Start is 425583 and end is 567444
Hello, I am 4
For id: 4. Start is 567444 and end is 709305
Hello, I am 7
For id: 7. Start is 993027 and end is 1134890
Hello, I am 2
For id: 2. Start is 283722 and end is 425583
Finished id: 7. Elapsed time: 0.019409
Hello, I am 5
For id: 5. Start is 709305 and end is 851166
Hello, I am 6
For id: 6. Start is 851166 and end is 993027
Finished id: 6. Elapsed time: 0.004363
Finished id: 5. Elapsed time: 0.021700
Finished id: 3. Elapsed time: 0.065244
Finished id: 2. Elapsed time: 0.077734
Finished id: 1. Elapsed time: 0.146834
Finished id: 4. Elapsed time: 0.152418
Finished id: 0. Elapsed time: 2.032460
Toc: 11548 seconds and 350684285 nanoseconds
Time elapsed (seconds): 2.048662
Total number of triangles: 3056386
```

(a) Load balancing pthreads, Scheduling: static, Threads: 8. Matrix com-Youtube.mtx. 10% speedup



```
----------Version 4----------
Tic: 12223 seconds and 621036786 nanoseconds
Toc: 12223 seconds and 930843929 nanoseconds
Time elapsed (seconds): 0.309807
Total number of triangles: 7978861

---------Version 4 Pthread is called---------
Tic: 12223 seconds and 930932289 nanoseconds
Current stack size -> 8388608
Hello, I am 0
For id: 0. Start is 0 and end is 65536
Hello, I am 7
For id: 7. Start is 458752 and end is 524288
Hello, I am 1
For id: 1. Start is 65536 and end is 131072
Hello, I am 2
For id: 2. Start is 131072 and end is 196608
Hello, I am 6
For id: 6. Start is 393216 and end is 458752
Hello, I am 5
For id: 5. Start is 327680 and end is 393216
Hello, I am 3
For id: 3. Start is 196608 and end is 262144
Hello, I am 4
For id: 4. Start is 262144 and end is 327680
Finished id: 7. Elapsed time: 0.097889
Finished id: 5. Elapsed time: 0.095962
Finished id: 0. Elapsed time: 0.116595
Finished id: 2. Elapsed time: 0.123036
Finished id: 6. Elapsed time: 0.126428
Finished id: 3. Elapsed time: 0.123485
Finished id: 1. Elapsed time: 0.134692
Finished id: 4. Elapsed time: 0.124316
Toc: 12224 seconds and 81524391 nanoseconds
Time elapsed (seconds): 0.150592
Total number of triangles: 7978861
```

(b) Load balancing pthreads. Scheduling: static, Threads: 8. Matrix rgg_n_2_19_s0.mtx. 50% speedup.

# 9 Notes

- Αναγνώριση χρονοβόρων διαδικασιών/επαναλήψεων
- Αφαίρεση εξαρτήσεων μεταξύ των επαναλήψεων (loop carrier dependency)
- Αναγνώριση βέλτιστου scheduling (κατανομή "βάρους")
- Binary vs Linear search sorted not very different

[1] Δεν έχει τρέξει στην συστοιχία αλλά σε προσωπικό σύστημα. FIX ME!