

Parallel and Distributed Systems

Vertexwise triangle counting

Θεόδωρος Κατζάλης

AEM:9282

katzalis@auth.gr

6/12/2020

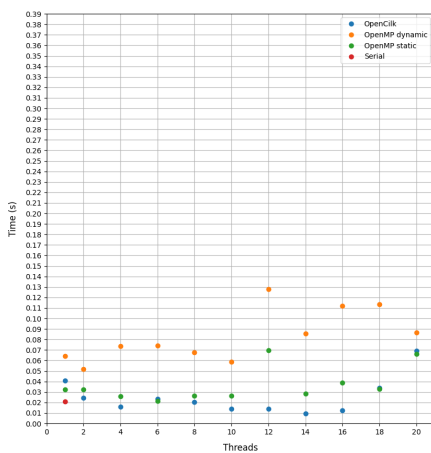
Contents

1	Introduction	1
2	$n = 1.441.295$, $m = 3.099.940$ (belgium_osm.mtx)	1
3	$n = 1.134.890$, $m = 5.975.248$ (com-Youtube.mtx)	2
4	$n = 326,186$, $m = 1,615,400$ (dblp-2010.mtx)	2
5	$n = 6.143$, $m = 1.227.742$ (mycielskian.mtx)	2
6	$n = 1.039.183$, $m = 6.229.636$ (NACA0015.mtx)	3
7	Conclusions	3

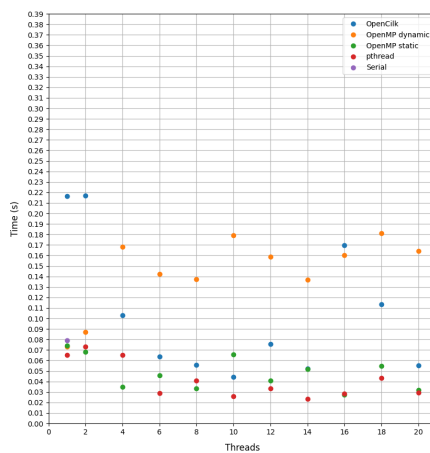
1 Introduction

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.

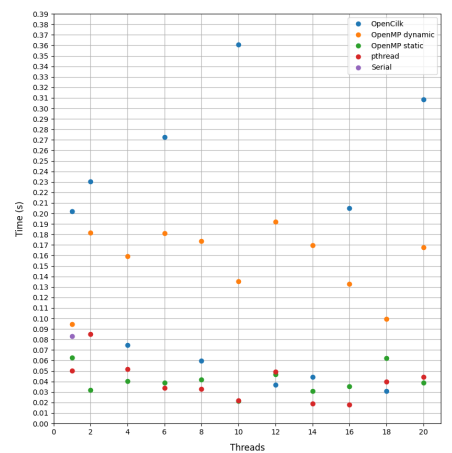
2 $n = 1.441.295$, $m = 3.099.940$ (belgium_osm.mtx)



(a) V3 binary search

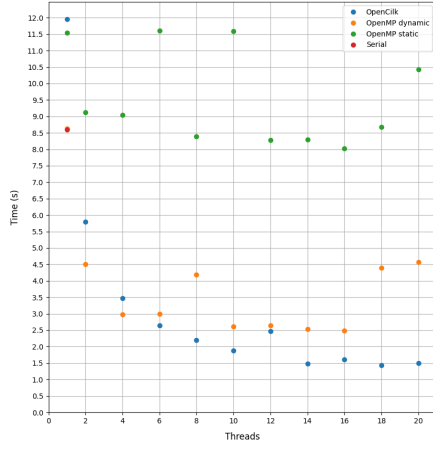


(b) V4 binary search

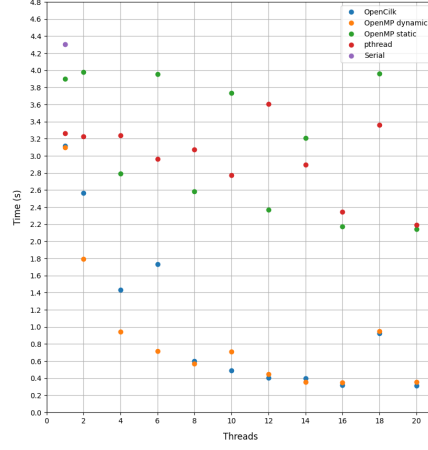


(c) V4 linear search

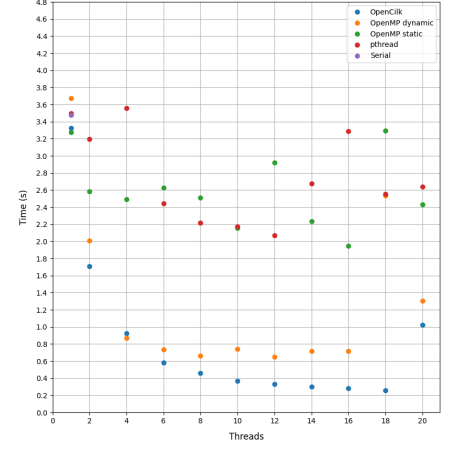
3 $n = 1.134.890$, $m = 5.975.248$ (com-Youtube.mtx)



(a) V3 binary search

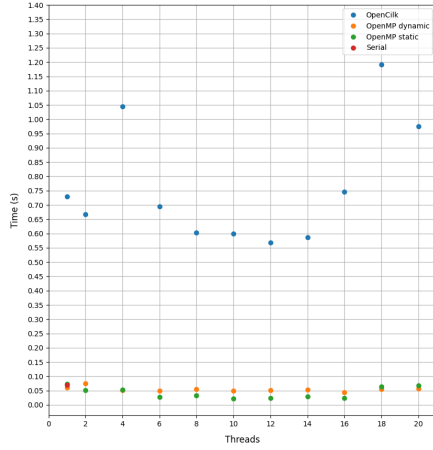


(b) V4 binary search

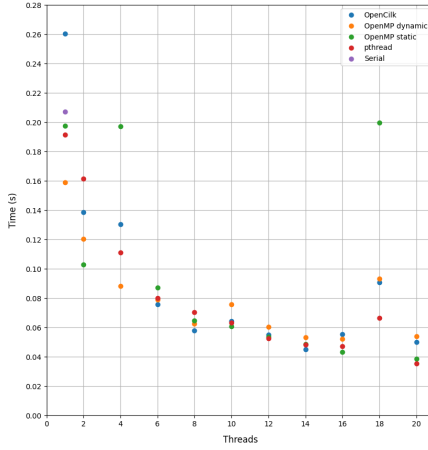


(c) V4 linear search

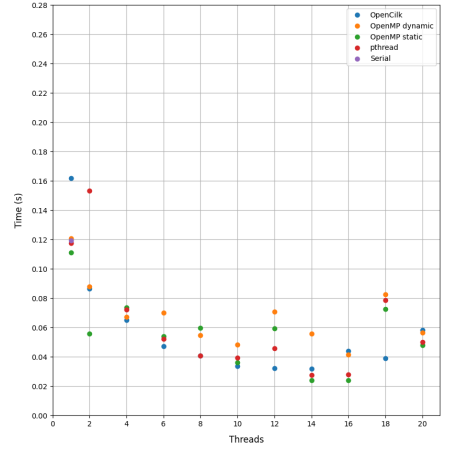
4 $n = 326,186$, $m = 1,615,400$ (dblp-2010.mtx)



(a) V3 binary search

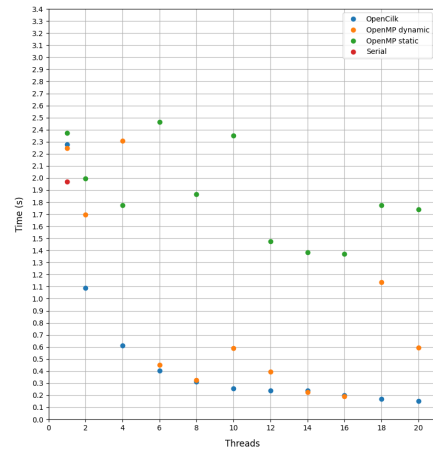


(b) V4 binary search

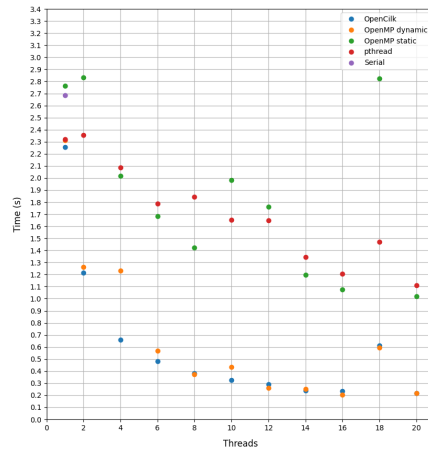


(c) V4 linear search

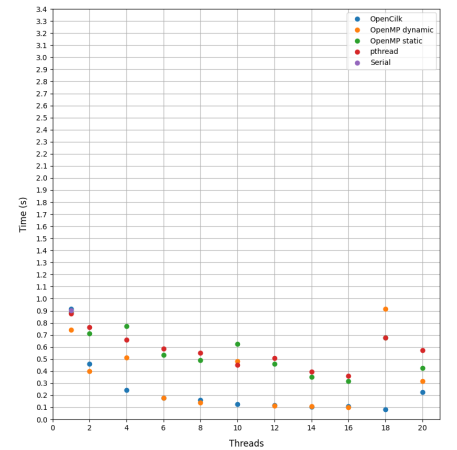
5 $n = 6.143$, $m = 1.227.742$ (mycielskian.mtx)



(a) V3 binary search

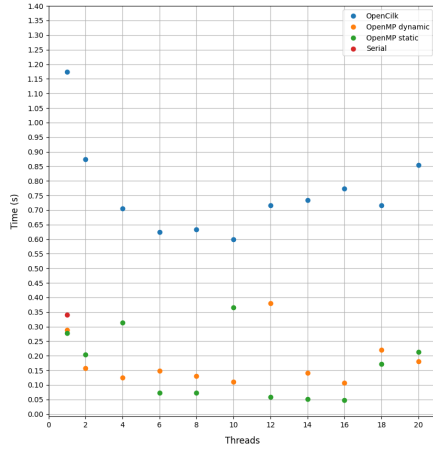


(b) V4 binary search

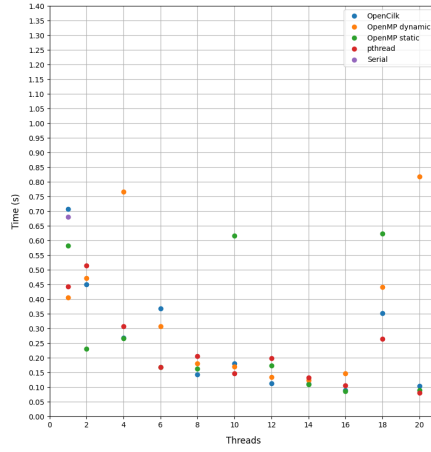


(c) V4 linear search

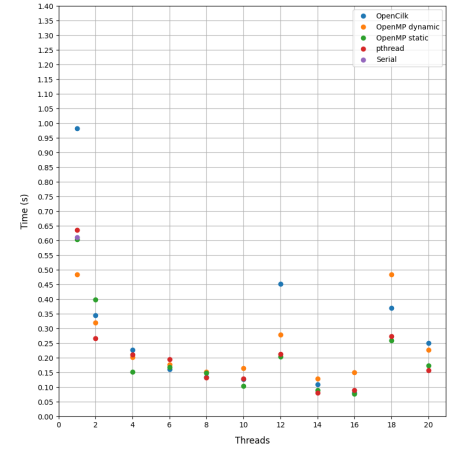
6 $n = 1.039.183$, $m = 6.229.636$ (NACA0015.mtx)



(a) V3 binary search



(b) V4 binary search



(c) V4 linear search

7 Conclusions

- “Οὐκ ἐν τῷ πολλῷ τὸ εὔ”. 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](#).