shared memory takes $\frac{k-1}{2}t_u$ µs in average, hence it directly depends on number of cores executing or waiting at the moment. Thus, the more cores are waiting, the more time it takes to acquire the lock and vice versa (the more time it takes to acquire the lock the more cores will be waiting). The vicious cycle leads to the collapse in the speed-up. This means that adding additional cores to the process can result in steep decrease in performance instead of increasing it, as it seen in Figure 1.2.

In Figure 1.1, at $t_x = 300$ µs, a small hump can be observed. This hump actually corresponds to the speed-up collapse. Figure 2.1 demonstrates five throughput plots according to t_x values of 100, 200, 300, 400 and 500µs. It can be clearly seen that the exact number of cores corresponding to the following collapse becomes higher as t_x increases.

As it can be seen in Figure 1.2 and Figure 1.3 whether ideal spinlock gives linear speed-up or not, clearly depends on the number of cores and parameters t_x, t_c , t_u. However, it is not a "perfectly" linear speed-up at any point. As it is demonstrated on the Figure 2.2, the population *N* is not constant, therefore the speed-up *n-N* apparently is not "perfectly" linear as *n*. Obviously, after certain point, increase in average number of cores either waiting or executing (N) becomes equal to increase in number of cores itself (n). At this point previously linear speed up hits the presumably never-ending plateau. According to approximation done as a part of this coursework the plateau occurs after number of cores reaches $n = \left\lceil \frac{t_x}{t_c+t_u} \right\rceil$ (the "ceiling" function. In this coursework, accuracies are 90.13% and 91.41% for $t_x = 100$ µs and $t_x = 300$ µs respectively). Actually, the equation of "speed up" converges to the value of $\frac{t_x}{t_c+t_u}$ (i.e. the plateau value). This can be done by simplifying the population function which should be expressed in terms of t_x, t_c, and t_u:

$$\lim_{n\to+\infty} N = \lim_{n\to+\infty} \frac{\sum_{k=1}^{n} k \frac{n!}{(n-k)!\, t_x^k}(t_c+t_u)^k}{\sum_{m=1}^{n} \frac{n!}{(n-m)!\, t_x^m}(t_c+t_u)^m} \Leftrightarrow \lim_{n\to+\infty} N = \lim_{n\to+\infty} \frac{ne^r - re^r}{e^r} = \lim_{n\to+\infty} (n-r), \text{ where } r = \frac{t_x}{t_c+t_u}.$$

In the model the t_c – the average time spent in the critical section should depend on number of cores waiting for the lock. This due to the fact, that cores actually cache not only 2 bytes of the spinlock but the whole cache line which is 64 bytes for Intel Core i3-i7. The owner core needs an exclusive access to the cache line in order to write a value to the surrounding data. However, the other cores constantly acquiring a shared access to the spinlock cache line will prevent the owning core from acquiring abovementioned exclusive access. This will result in cache miss for every write operation performed by owning core to the spinlock cache line. The throughput can be reduced by a factor of two for k=1 and by a factor of 10 for k>1 (Corbet, J. 2013) [1].

Also, another model is that one core can be reserved as a dispatcher, passing inter-core messages and managing queues. However, the bigger gets number of managed critical sections the more complicated and "slow" gets the model due to the very limited cache size.

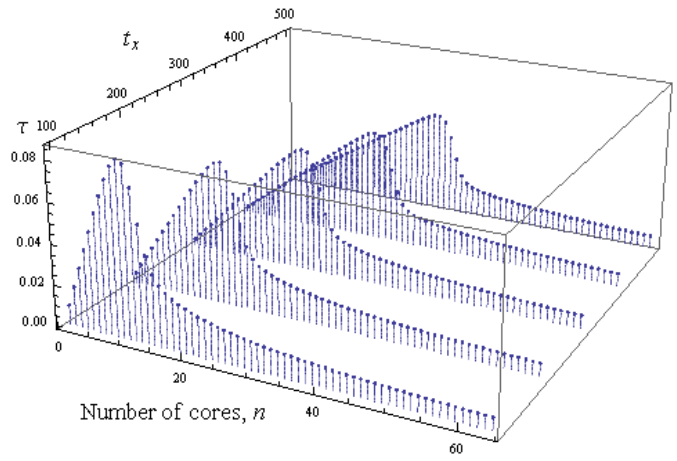

Figure 2.1 Throughput τ

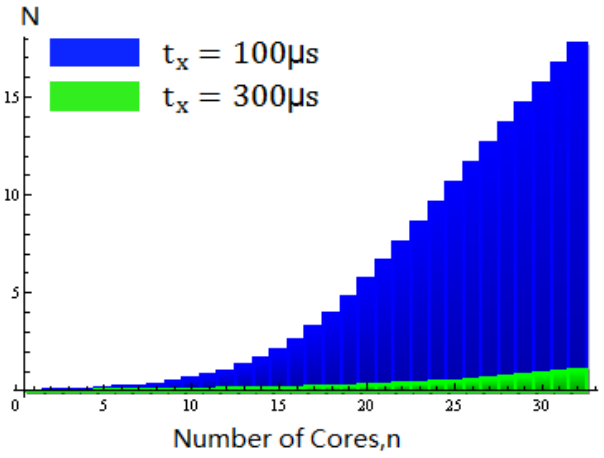

Figure 2.2, Population, N (ideal-lock)

[1] Jonathan Corbet, (2013), *"Improving ticket spinlocks"*. Available at: http://lwn.net/Articles/531254/. Retrieved on 2013.10.29