Peculiarities of optimization for multicore processors

By David Livshin, published on October 24, 2018

In this article we will compare benefits of optimizations applied to a scalar code vs the same scalar optimizations applied to the parallelized version of the scalar code. First we provide an example to demonstrate what is happening and then provide formal explanation.

Example

Assume that sequential code

- runs for 100 units of time – *uot* – e.g. 100 hours
- can be optimized leading 20% execution time improvement
- the whole task can be successfully parallelized on 4-core processor

This leads to the following metrics to be analyzed:

1. ***ES*** execution time of the original sequential code – 100*uot*
2. ***EOS*** execution time of the optimized sequential code – 80*uot*
3. ***EP*** execution time of the parallelized code – 25*uot*
4. ***EPO*** execution time of the parallelized and optimized code – 20*uot*

The following table summarizes the above data and provides improvements of the optimized/parallelized code over original sequential code.

	ES	***EOS***	***EP***	***EPO***
Execution times	100	80	25	20
Improvement over ***ES***		20%	75%	80%

What becomes obvious that opimization of the sequential code (originaly 20%) when combined with parallelization got diluted to 5% only: ***EPO - EP***!

Formal Explanation

We denote the improvements archived by the letter ש, (pronounced **shin**, from Hebrew **שיפור** − improvement) and define it to be

$$ש = \frac{NotOptimzedExecutionTime - OptimizedExecutionTime}{NotOptimzedExecutionTime}$$

where

NotOptimizedExecutionTime is execution time of not optimized code

OptimizedExecutionTime is execution time of the optimized code

Note that

$$0 \leqslant \varpi < 1$$

and the execution of the Optimized code that was improved by ϖ is equal

OptimizedExecutionTime = (1 - ϖ)****NotOptimizedExecutionTime***

or, making ***NotOptimizedExecutionTime*** to be 1

OptimizedExecutionTime = 1 - ϖ

We apply our optimizations – sequential code optimizations and code parallelization – to the portion of the code, percentage of execution time of which we denote by ***p*** (assume $\boldsymbol{p} \neq 0$); we will all so refer to this section of code by the same name. Then the execution time of the program is equal to

$$(1- \boldsymbol{p}) + \boldsymbol{p}$$

(1- ***p***) being execution time of unoptimizable section of the code and ***p*** being execution time of the section of the code that will be optimized.
Denote ϖ_{s} to be the total improvement achieved when applying sequential code optimizations to the section ***p*** of the code and denote ϖ_{sLocal} to be the improvement achieved when applying sequential code optimizations to the section ***p*** of the code.

Thus after applying sequential code optimizations to the section ***p*** of the code the execution time of the optimzed code will be

$$1 - \varpi_{\mathrm{s}} = (1 – \boldsymbol{p}) + (1 – \varpi_{\mathrm{sLocal}})*\boldsymbol{p}$$

from which we can derive

$$\varpi_{\mathrm{sLocal}} = \frac{\varpi_s}{\boldsymbol{p}}$$

For example if 30% is the total improvements achieved when applying code optimizations to the sequential code (ϖ_{s}=.3) that takes 70% of time to execute (***p*** = .7), then the execution time of the optimized code will be 1 - .3 and the improvements achieved when applying sequential code optimizations to the section ***p*** of the code shall be $\frac{.3}{.7}$.

After applying parallelization to the section ***p*** of the code the total execution time of the parallelized code (note that only ***p*** was parallelized) will be (ignoring overhead of doing parallelization)

$$1 - \varpi_p = (1 – \boldsymbol{p}) + \frac{\boldsymbol{p}}{\boldsymbol{NC}}$$

where ***NC*** is the number of cores (threads) utilized.
After applying sequential optimization to the thread of parallelized section ***p*** of the code the execution time of the code will be (provided this can be done which is often the case)

$$1 - \varpi_{p+s} = (1 – \boldsymbol{p}) + (1 – \varpi_{\mathrm{sLocal}})* \frac{\boldsymbol{p}}{\boldsymbol{NC}} = (1 – \boldsymbol{p}) + (1 – \frac{\varpi_s}{\boldsymbol{p}})* \frac{\boldsymbol{p}}{\boldsymbol{NC}}$$

From the above established formulas we can derive some important metrics.

How much sequential optimizations improve the parallelized code

To find out that, we need to calculate the difference between ϖ_{p+s} and ϖ_p.

$$\varpi_{p+s} - \varpi_p = (1 - \varpi_p) - (1 - \varpi_{p+s}) = (1 - \boldsymbol{p}) + \frac{\boldsymbol{p}}{\boldsymbol{NC}} - (1 - \boldsymbol{p}) - \left(1 - \frac{\varpi_s}{\boldsymbol{p}}\right) * \frac{\boldsymbol{p}}{\boldsymbol{NC}}$$

or

$$\varpi_{p+s} - \varpi_p = \frac{\varpi_s}{\boldsymbol{NC}}$$

Note that the result we obtained doesn't depend on $\boldsymbol{p}$.

However lets find the difference as a function of sequential code improvement for the section $\boldsymbol{p}$ to be optimized (as a function of ϖ_{sLocal}); we get

$$\varpi_{p+s} - \varpi_p = \frac{\varpi_{sLocal} * \boldsymbol{p}}{\boldsymbol{NC}}$$

How much sequential optimizations improve the execution time of the parallelized code

The execution time of the parallelized code is

$$1 - \varpi_p = (1 - \boldsymbol{p}) + \frac{\boldsymbol{p}}{\boldsymbol{NC}}$$

The execution time of the parallelized code after applying sequential optimizations is

$$1 - \varpi_{p+s} = (1 - \boldsymbol{p}) + (1 - \varpi_{\text{sLocal}}) * \frac{\boldsymbol{p}}{\boldsymbol{NC}}$$

Therefore the sequential optimizations improve the execution time of the parallelized code by

$$\frac{\dfrac{\varpi_{sLocal} * \boldsymbol{p}}{\boldsymbol{NC}}}{(1 - \boldsymbol{p}) + \dfrac{\boldsymbol{p}}{\boldsymbol{NC}}} = \frac{\varpi_{sLocal} * \boldsymbol{p}}{\boldsymbol{NC} * (1 - \boldsymbol{p}) + \boldsymbol{p}}$$

Thus applying sequential optimizations to code that was parallelized does improve it execution time.

How much adding sequential optimizations to parallelized code improve the total execution time

The total execution time of unoptimized code is

$$(1 - \boldsymbol{p}) + \boldsymbol{p} = 1$$

The execution time of the parallelized code after applying sequential optimizations is

$$1 - \varpi_{p+s} = (1 - \boldsymbol{p}) + (1 - \varpi_{\text{sLocal}}) * \frac{\boldsymbol{p}}{\boldsymbol{NC}}$$

Therefore adding sequential optimizations to parallelized code improve the total execution time by

$$\boldsymbol{p} * \left(1 + \frac{\varpi_{sLocal} - 1}{\boldsymbol{NC}}\right)$$

which clearly shows that contribution of sequential optimization getting diluted as the number of threads grows.

Conclusions

It seems that the effect of sequential code optimizations, when applied to the a parallel section of the code, is diluted by the number of threads involved: *the more threads are utilized the less the sequential optimizations affect the overall code performance*. All the effort in establishing and implementing these optimizations seems to become obsolete on the multicore systems. So, is the time over for using these technologies?

Obviously multicore systems require new algorithms to be developed. But the time of sequential code optimizations is not over yet. With apologies to Mark Twain, the report of demise of sequential code optimizations is greatly exaggerated! And that is because:

1. Not every code sequence may be parallelized
2. In order to parallelize sequential code, it is often necessary to apply sequential code optimizations for it to become “parallelizable”; we learned this while developing Dalsoft’s auto-parallelizer.
3. Sequential code optimizations do improve overall code performance of parallelized code.