Calculating number of cores to benefit from parallelization

By David Livshin, published on November 15, 2018

david.livshin@dalsoft.com

In this article we will explain one of the internal mechanisms utilized by ***dco*** (see this) to decide if the given code can benefit from parallelization.

While creating parallel version of a sequential code, ***dco*** creates number of code segments, such as:

- code to calculate loop count
- code to perform dynamic data flow analysis (memory dependencies)
- code to setup data necessary for threads execution (entry values)

Note that “code to calculate loop count” and “code to perform dynamic data flow analysis” is executed by the master process but “code to setup data necessary for threads execution” is executed inside a thread.

Denote:

PET to be program execution time
PPET to be execution time of the parallelized program
LC to be a loop count
NT to be number of threads (cores) utilized
CLC overhead of the code to calculate loop count
CMD overhead of the code to perform data flow analysis
CEV overhead of the all iterations of the code to calculate entry values
SCO static code overhead – mostly code to spawn/terminate team of threads
TLC to be a loop count inside thread

Note that ***PET***, ***CLC***, ***CMD*** and ***CEV*** are values for execution of ***LC*** iterations; ***SCO*** is executed only once. For every item ***ABC*** values for execution of ***LC*** iterations denote the execution time for one iteration as ABC_1, thus PET_1 is execution time of one loop iteration of the program and $PET_1 = \frac{PET}{LC}$. Usually PET_1, CLC_1, CMD_1, CEV_1 and ***SCO*** can be calculated as numeric values at the time of generation of parallel code (parallelization).

The loop count inside a thread is equal (approximately) to:

$$TLC = \frac{LC}{NT}$$

Inside every thread, the code generated by ***dco***, before entering the main loop, calculates data necessary for threads execution (entry values). This is done by executing generated code segment for number of iterations equal to the sum of iterations for the threads with preceding

numbers. All this is done in parallel, therefore only the thread that executes most iterations counts. Because this number of iterations increases for with the number of the thread being analyzed, the largest number will be for the last thread. For the last thread the number of the preceding threads is equal to $NT - 1$, each preceding thread being executed by TLC iterations which bring the total number of iterations to execute the generated segment that setup data necessary for threads execution to be

$$(NT-1)*TLC$$

Overhead of the all LC iterations of the code to calculate entry values is equal to CEV, therefore overhead of one iteration of the code to calculate entry values is equal to $\frac{CEV}{LC}$ so overhead of the of the code to calculate entry values for the last thread is

$$(NT-1)*TLC*\frac{CEV}{LC} = (NT-1)*\frac{LC}{NT}*\frac{CEV}{LC} = \frac{(NT-1)}{NT}*CEV$$

Inside every thread the execution time of the main loop is

$$\frac{PET}{NT}$$

Thus the execution time of the parallelized program is

$$PPET = CLC + CMD + SCO + \frac{(NT-1)}{NT}*CEV + \frac{PET}{NT}$$

or

$$PPET = CLC + CMD + CEV + CSO + \frac{PET-CEV}{NT}$$

For parallelization to be beneficial it must be

$$PPET < PET$$

or

$$CLC + CMD + CEV + CSO + \frac{PET-CEV}{NT} < PET$$

or

$$\frac{PET-CEV}{NT} < PET - (CLC + CMD + CEV + CSO)$$

Code to calculate entry values is part of the program to be executed, thus

$CEV <= PET$ and therefore $(PET - CEV) >= 0$

Which make it clear that if $(PET - (CLC + CMD + CEV + CSO)) <= 0$ parallelization will not be beneficial.

If $(PET - (CLC + CMD + CEV + CSO)) > 0$ and thus $(PET - CEV) > 0$ then for parallelization to be beneficial it must be

$$NT > \frac{PET-CEV}{PET-(CLC+CMD+CEV+CSO)}$$

or

$$NT > 1 + \frac{CLC+CMD+SCO}{PET-(CLC+CMD+CEV+SCO)}$$

Substituting the values for the single iterations of the items, for parallelization to be beneficial:

$$(PET_1 - (CLC_1 + CMD_1 + CEV_1 + \frac{SCO}{LC})) > 0 \quad (1)$$

and

$$NT > \frac{PET_1 - CEV_1}{PET_1 - \left(CLC_1 + CMD_1 + CEV_1 + \frac{CSO}{LC}\right)} \quad (2)$$

or

$$NT > 1 + \frac{CLC_1 + CMD_1 + \frac{SCO}{LC}}{PET_1 - \left(CLC_1 + CMD_1 + CEV_1 + \frac{SCO}{LC}\right)} \quad (3)$$

Note that according to (2), ***NT*** is inversely proportional to ***LC:*** the more iteration there are in the loop the fewer threads are necessary to benefit from parallelization. So to calculate the minimal number of threads for parallelization to be beneficial for some number of iterations assume ***LC*** in (2) to be so large as to make $\frac{SCO}{LC}$ negligible. Thus the minimal number of threads must exceed:

$$1 + \frac{CLC_1 + CMD_1}{PET_1 - \left(CLC_1 + CMD_1 + CEV_1\right)} \quad (4)$$

Now let calculate the minimal number ***LC*** of loop iteration necessary in order to make parallelization beneficial for a given number ***NT*** of threads. According to (1), (2) may be rewritten as

$$PET_1 - \left(CLC_1 + CMD_1 + CEV_1 + \frac{CSO}{LC}\right) > \frac{PET_1 - CEV_1}{NT}$$

or

$$\frac{CSO}{LC} < PET_1 - \left(CLC_1 + CMD_1 + CEV_1\right) - \frac{PET_1 - CEV_1}{NT}$$

or

$$\frac{CSO}{LC} < \left(PET_1 - CEV_1\right) * \left(1 - \frac{1}{NT}\right) - \left(CLC_1 + CMD_1\right)$$

Note that $\frac{CSO}{LC} > 0$ and therefore $\left(PET_1 - CEV_1\right) * \left(1 - \frac{1}{NT}\right) - \left(CLC_1 + CMD_1\right) > 0$

thus

$$LC > \frac{SCO}{\left(PET_1 - CEV_1\right) * \left(1 - \frac{1}{NT}\right) - \left(CLC_1 + CMD_1\right)} \quad (5)$$

For example, if PET_1 = 100(clocks), CLC_1 = 20, CMD_1 = 40, CEV_1 = 10 and ***SCO*** = 1000 then:

in order for parallel version of the code to be faster than original sequential input we need:
using (4)

$$NT > 1 + \frac{20 + 40}{100 - \left(20 + 40 + 10\right)} = 1 + 2$$

at least 4 threads.

for $\boldsymbol{LC}$ = 10000

using (3)

$$\boldsymbol{NT} > 1 + \frac{20+40+0.1}{100-(20+40+10+0.1)} = 1 + \frac{60.1}{29.9} = 1 + 2.01$$

at least 4 threads.

for $\boldsymbol{LC}$ = 100

using (3)

$$\boldsymbol{NT} > 1 + \frac{20+40+10}{100-(20+40+10+10)} = 1 + \frac{70}{20} = 1 + 3.5$$

at least 5 threads.

for $\boldsymbol{LC}$ = 10

using (1)

$$(\boldsymbol{PET_1} - (\boldsymbol{CLC_1} + \boldsymbol{CMD_1} + \boldsymbol{CEV_1} + \frac{\boldsymbol{SCO}}{\boldsymbol{LC}})) = (100 - (20 + 40 + 10 + 100)) < 0$$

there is no way to benefit from parallelization.

for $\boldsymbol{NT}$ = 4

using (5)

$$\boldsymbol{LC} > \frac{1000}{(100-10)*(1-\frac{1}{4})-(20+40)} = \frac{1000}{67.5-60} = 133.3333$$

at least 134 iterations.

Conclusions

The decision if the given code can benefit from parallelization is far from being intuitive and require subtle and complicated analysis. It is possible that the answer will depend on number of cores available for parallelization.