

# When OpenMP fails to parallelize the code

By David Livshin, published on August 22, 2019

Copyright © 2019 David Livshin. All rights reserved.  
david.livshin@dalsoft.com

OpenMP is a very powerful tool for creation of parallel code. However OpenMP requires user to be responsible for everything in the processors of the parallel code generation, including:

- specifying the code to be parallelized with OpenMP being very particular what code it accepts for processing
- providing proper parameters that guide the parallelization
- guaranteeing that parallelization is possible e.g. checking for race conditions

Also there are code patterns that OpenMP is not capable of treating.

We conceived and implemented a software tool that may help to overcome the problems cited above. We developed an optimizing/parallelizing code system for the x86 family of processors. *dco* ( see [this](#) ) is a software package specifically designed to auto-parallelize x86 assembly code. *dco* may be used to treat the code that OpenMP is not able of treating as well as to relieve the user from preparing the code to be treated by OpenMP.

All the timing cited in the following text refers to the execution of the code based on the corresponding example and was obtained by executing on the [Microway's](#) compute cluster utilizing

Intel(R) Xeon(R) Silver 4110 CPU @2.10GHz **16 cores**

processor.

## using *dco* to parallelize the code not treatable by OpenMP

### not structured block

OpenMP treats only “structured blocks” ( quote from the OpenMP spec ): *The block of code must be structured in the sense that it must have a single point of entry at the top and a single point of exit at the bottom; i.e., branching in or out of a structured block is not allowed.*

*dco* can handle code with multiple points of exit. The following code illustrates that:

```
double NotStructuredBlock( double a[], unsigned int cnt, unsigned
int *indx )
{
double ret;
```

```

unsigned int j;

ret = 0.;
for ( j = 0; j < cnt; j++ )
{
    if ( a[j] > 0.1 )
    {
        break;
    }
    ret += exp( sin( a[j] ) );
}

*indx = j;

return ret;
}

```

The above code is not structured ( the line `break;` makes it a such ) and therefore may not be parallelized by OpenMP. The above loop was successfully parallelized by *dco*.

The original serial code run for **1.286** seconds, the parallel version of the code run for **0.639** seconds thus providing **50%** improvement.

## not a countable loop

Consider the following code:

```

double foo( double x, unsigned int cnt )
{
    double x1, ret;
    unsigned int i;

    ret = 0.;
    for ( i = 0; i < cnt; )
    {
        x1 = 2.0 * x - 1;
        if ( x1 < 1.0 )
        {
            x1 = exp( x1 );
            i = i + 3;
            x = x * 2.;
        }
    }
}

```

```

else if ( x1 < 2.0 )
{
    x1 = sin( x1 );
    i = i + 1;
    x += 1.;
}
else
{
    x1 = cos( x1 );
    i = i + 2;
    x = x/2.;
}

ret += x1;
}

return ret;
}

```

Note that this loop does not conform to a requirement of OpenMP – loop must be “countable” - and therefore may not be processed by OpenMP; however the above loop was successfully parallelized by *dco*.

The original serial code run for **1.387** seconds, the parallel version of the code run for **1.133** seconds thus providing **18%** improvement.

## not parallel on OpenMP - data recurrence

The following is example of a recurrence and may not be directly parallelized by OpenMP:

```

double a[ARRAY_DIM] , c[ARRAY_DIM] ;
.
double wrap;

wrap = 0.;
for ( i = 0; i < ARRAY_DIM; i++ ) {
    c[i] = exp( sin( wrap ) );
    wrap += a[i];
}

```

the above loop was successfully parallelized by *dco*.

The original serial code run for **1.841** seconds, the parallel version of the code run for **1.019** seconds thus providing **45%** improvement.

## using *dco* to parallelize the code that is treatable by OpenMP

Consider EP - "Embarrassingly Parallel" suite from the NAS Parallel Benchmarks ( NPB ).

```
do 140 i = 1, nk
  x1 = 2.d0 * x(2*i-1) - 1.d0
  x2 = 2.d0 * x(2*i) - 1.d0
  t1 = x1 ** 2 + x2 ** 2
  if (t1 .le. 1.d0) then
    t2 = sqrt(-2.d0 * log(t1) / t1)
    t3 = (x1 * t2)
    t4 = (x2 * t2)
    l = max(abs(t3), abs(t4))
    q(l) = q(l) + 1.d0
    sx = sx + t3
    sy = sy + t4
  endif
140 continue
```

The code, as it is presented in NPB serial suite, is not parallelizable. The line

```
q(l) = q(l) + 1.d0
```

introduces data dependency.

OpenMP offers number of ways to solve this dependency and generate parallel code. However not all of them produce optimal parallel code and all require alteration of the original source code. *dco* doesn't require user to change the source code and performs all necessary alterations generating parallel code "automatically".

See [this](#) for execution data regarding this benchmark.

## Conclusions

We showed the cases *dco* successfully parallelizing the code that OpenMP is not capable of treating. Even if OpenMP able to process the code, *dco* still may be useful offering easy and automatic way to create a parallel version of the input source.