Collapse Clause

- Loop
-

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice and Octave.

Clauses (7)

collapse (**n**)

allows you to parallelize multiple loops in a nest
without introducing nested parallelism.

Only one **collapse** clause is allowed
on a worksharing **for** or **parallel for** pragma.

n : the number of nested loops to be parallelized

the specified number of loops must be present lexically.
that is, none of the loops can be in a called subroutine.

https://www.ibm.com/docs/en/xl-c-aix/13.1.2?topic=processing-pragma-omp-section-pragma-omp-sections

Clauses (8)

The loops must form a rectangular iteration space and
the bounds and stride of each loop
must be invariant over all the loops.

If the loop indices are of different size,
the index with the largest size
will be used for the collapsed loop.

The loops must be perfectly nested;
that is, there is no intervening code
nor any OpenMP pragma
between the loops which are collapsed.

https://www.ibm.com/docs/en/xl-c-aix/13.1.2?topic=processing-pragma-omp-section-pragma-omp-sections

Clauses (9)

The associated do-loops must be structured blocks.
Their execution must not be terminated by an break statement.

If multiple loops are associated to the loop construct,
only an iteration of the innermost associated loop
may be curtailed by a continue statement.

If multiple loops are associated to the loop construct,
there must be no branches to any of the loop termination statements
except for the innermost associated loop.

https://www.ibm.com/docs/en/xl-c-aix/13.1.2?topic=processing-pragma-omp-section-pragma-omp-sections

Collapse example (1)

The **collapse** clause is used to convert a prefect nested loop into a single loop then parallelize it.

```
#include <stdio.h>
#include <omp.h>

int main(void)
{
    #pragma omp parallel for
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            printf("Thread number is %d\n", omp_get_thread_num());
        }
    }

    return 0;
}
```

```
# gcc -fopenmp parallel.c
# ./a.out
Thread number is 0
Thread number is 0
Thread number is 0
Thread number is 0
Thread number is 0
Thread number is 3
Thread number is 3
Thread number is 3
Thread number is 3
Thread number is 3
Thread number is 1
Thread number is 1
Thread number is 1
Thread number is 1
Thread number is 1
Thread number is 2
Thread number is 2
Thread number is 2
Thread number is 2
Thread number is 2
```

https://nanxiao.gitbooks.io/openmp-little-book/content/posts/collapse-clause.html?q=

Collapse example (2)

Every iteration of outer loop will be dispatched to one thread to run:

```
#pragma omp parallel for
for (int i = 0; i < 4; i++)
{
    for (int j = 0; j < 5; j++)
    {
        printf("Thread number is %d\n", omp_get_thread_num());
    }
}
```

Each thread will execute the inner loop sequentially:

So there are only 4 threads in active state actually.

https://nanxiao.gitbooks.io/openmp-little-book/content/posts/collapse-clause.html?q=

Collapse example (3)

```c
#include <stdio.h>
#include <omp.h>

int main(void)
{
    #pragma omp parallel for  collapse(2)
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            printf("Thread number is %d\n", omp_get_thread_num());
        }
    }

    return 0;
}
```

```
# gcc -fopenmp parallel.c
# ./a.out
Thread number is 0
Thread number is 2
Thread number is 18
Thread number is 16
Thread number is 6
Thread number is 8
Thread number is 7
Thread number is 10
Thread number is 14
Thread number is 12
Thread number is 13
Thread number is 17
Thread number is 15
Thread number is 9
Thread number is 11
Thread number is 19
Thread number is 4
Thread number is 3
Thread number is 5
Thread number is 1
```

https://nanxiao.gitbooks.io/openmp-little-book/content/posts/collapse-clause.html?q=

Collapse example (4)

This time we can see 20 threads are utilized.

The integer argument of **collapse** (i.e., 2 in this example) identifies
how many loops to be parallelized,
and counted from outer side to inner side

Please be aware that **collapse(1)** and no collapse
take the same effect for loop parallelism

https://nanxiao.gitbooks.io/openmp-little-book/content/posts/collapse-clause.html?q=

Clauses (10)

Use the OpenMP **collapse** clause
to increase the total number of iterations
that will be partitioned across the available number of OMP threads
by reducing the granularity of work to be done by each thread.

If the amount of work to be done by each thread
is non-trivial (after collapsing is applied),
this may improve the parallel scalability of the OMP application.

https://software.intel.com/content/www/us/en/develop/articles/openmp-loop-collapse-directive.html

Clauses (11)

You can improve performance by avoiding
use of the collapsed-loop indices (if possible)
inside the collapse loop-nest

since the compiler has to recreate them
from the collapsed loop-indices
using divide/mod operations AND

the uses are complicated enough
that they don't get dead-code-eliminated
as part of compiler optimizations

https://software.intel.com/content/www/us/en/develop/articles/openmp-loop-collapse-directive.html

Clauses (12)

```
#pragma omp parallel for collapse(2)
  for (i = 0; i < imax; i++) {
    for (j = 0; j < jmax; j++) a[ j + jmax*i] = 1.;
  }
```

Modified example for better performance:

```
#pragma omp parallel for collapse(2)
  for (i = 0; i < imax; i++) {
     for (j = 0; j < jmax; j++) a[ k++] = 1.;
  }
```

https://software.intel.com/content/www/us/en/develop/articles/openmp-loop-collapse-directive.html

References

[1] en.wikipedia.org

[2] M Harris, http://beowulf.lcs.mit.edu/18.337-2008/lectslides/scan.pdf