OpenMP: For & Scheduling

Jun 13, 2016

OpenMP specialty is a parallelization of loops. The loop construct enables the parallelization.

```
#pragma omp parallel for
for (...)
{ ... }
```

OpenMP then takes care about all the details of the parallelization. It creates a team of threads and distributes the iterations between the threads.

In this article, we explore how OpenMP schedules the iterations between the threads and how can we change this behavior. First, we examine different ways to specify a scheduling type of a loop. Then, we describe five different scheduling types.

Scheduling

We describe how a programmer can determine a scheduling type of a loop.

Explicit

If the loop construct has explicit `schedule` clause

```
#pragma omp parallel for schedule(scheduling-type)
for (...)
{ ... }
```

then OpenMP uses `scheduling-type` for scheduling the iterations of the for loop.

Runtime

If the `scheduling-type` (in the schedule clause of the loop construct) is equal to `runtime` then OpenMP determines the scheduling by the internal control variable `run-sched-var`. We can set this variable by setting the environment variable `OMP_SCHEDULE` to the desired scheduling type. For example, in `bash`-like terminals, we can do

```
$ export OMP_SCHEDULE=sheduling-type
```

Another way to specify `run-sched-var` is to set it with `omp_set_schedule` function.

```
...
omp_set_schedule(sheduling-type);
...
```

Default

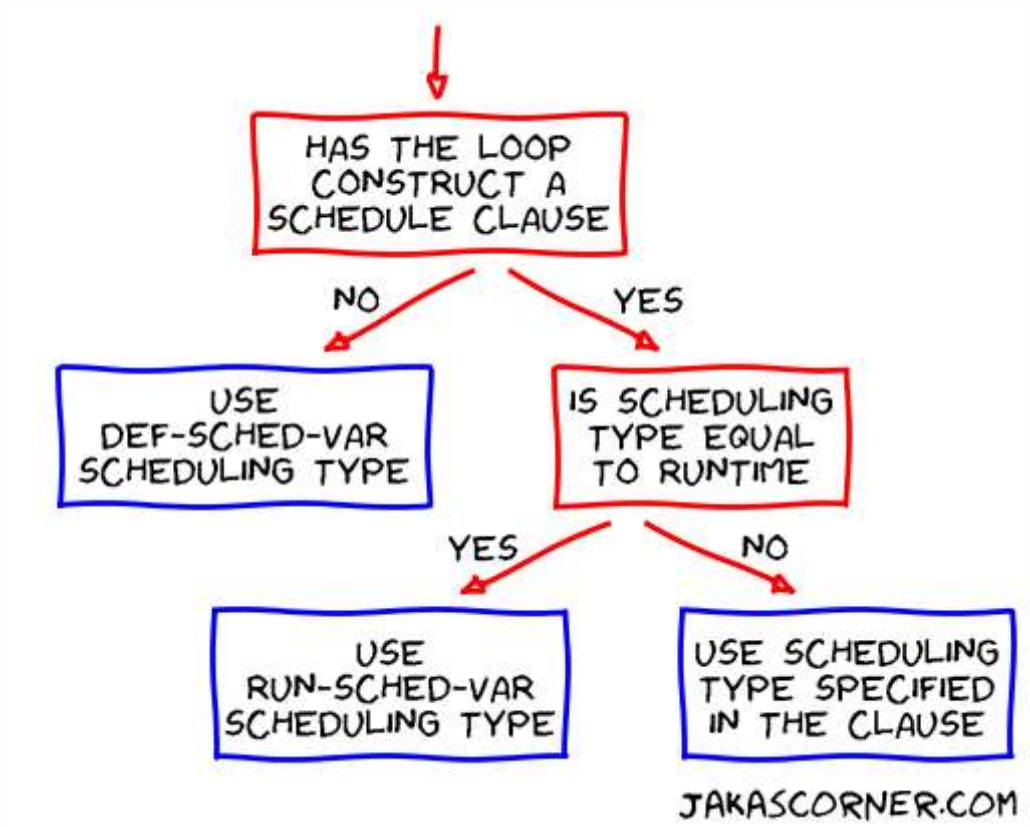
If the loop construct does not have an explicit `schedule` clause

```
#pragma omp parallel for
for (...)
{ ... }
```

then OpenMP uses the default scheduling type. It is defined by the internal control variable `def-sched-var` and it is implementation dependent.

Summary

The following figure summarizes how OpenMP determines a scheduling type of a for loop.



The scheduling types

We can choose between five different scheduling types:

- `static`,
- `dynamic`,
- `guided`,
- `auto` and
- `runtime`.

Static

The `schedule(static, chunk-size)` clause of the loop construct specifies that the for loop has the static scheduling type. OpenMP divides the iterations into chunks of size `chunk-size` and it distributes the chunks to threads in a circular order.

When no `chunk-size` is specified, OpenMP divides iterations into chunks that are approximately equal in size and it distributes at most one chunk to each thread.

Here are three examples of static scheduling.

```
schedule(static):
****************
                ****************
                                ****************
                                                ****************
```

```
schedule(static, 4):
****            ****            ****            ****
    ****            ****            ****            ****
        ****            ****            ****            ****
            ****            ****            ****            ****
```

```
schedule(static, 8):
********                        ********
        ********                        ********
                ********                        ********
                        ********                        ********
```

Let me explain the examples. We parallelized a for loop with 64 iterations and we used four threads to parallelize the for loop. Each row of stars in the examples represents a thread.

Each column represents an iteration.

The first example (`schedule(static)`) has 16 stars in the first row. This means that the first tread executes iterations 1, 2, 3, …, 15 and 16. The second row has 16 blanks and then 16 stars. This means that the second thread executes iterations 17, 18, 19, …, 31, 32. Similar applies to the threads three and four.

We see that for `schedule(static)` OpenMP divides iterations into four chunks of size 16 and it distributes them to four threads. For `schedule(static, 4)` and `schedule(static, 8)` OpenMP divides iterations into chunks of size 4 and 8, respectively.

The static scheduling type is appropriate when all iterations have the same computational cost.

Dynamic

The `schedule(dynamic, chunk-size)` clause of the loop construct specifies that the for loop has the dynamic scheduling type. OpenMP divides the iterations into chunks of size `chunk-size`. Each thread executes a chunk of iterations and then requests another chunk until there are no more chunks available.

There is no particular order in which the chunks are distributed to the threads. The order changes each time when we execute the for loop.

If we do not specify `chunk-size`, it defaults to one.

Here are four examples of dynamic scheduling.

```
schedule(dynamic):
*   ** **  * * *  *      *  *    **   *  *  * *       *  *   *
  *       *     *    * *     * *   *    *        * *   *    *
 *       *    *     * *   *   *     *  *      *  *  *  *  *   *
   *  *     *    * *    *  *    *    *    ** *  *   *     *   *
```

```
schedule(dynamic, 1):
    *    *     *        *   *    * *  *  *         *  * *  * *
*  *  *   * *     *  * * *    * *      *   ***  *   *         *
 *   *  *  *  *    ** *    *      *  *  * *   *  *   *   *
  *    *     *  **        *  * *    *          *  *    *  * *  *
```

```
schedule(dynamic, 4):
            ****                    ****                    ****
****            ****    ****            ****        ****
```

```
    ****            ****    ****            ****        ****
        ****                    ****            ****
```

```
schedule(dynamic, 8):
               ********                                ********
                       ********        ********
********                       ********        ********
        ********
```

We can see that for `schedule(dynamic)` and `schedule(dynamic, 1)` OpenMP determines similar scheduling. The size of chunks is equal to one in both instances. The distribution of chunks between the threads is arbitrary.

For `schedule(dynamic, 4)` and `schedule(dynamic, 8)` OpenMP divides iterations into chunks of size four and eight, respectively. The distribution of chunks to the threads has no pattern.

The dynamic scheduling type is appropriate when the iterations require different computational costs. This means that the iterations are poorly balanced between each other. The dynamic scheduling type has higher overhead then the static scheduling type because it dynamically distributes the iterations during the runtime.

Guided

The guided scheduling type is similar to the dynamic scheduling type. OpenMP again divides the iterations into chunks. Each thread executes a chunk of iterations and then requests another chunk until there are no more chunks available.

The difference with the dynamic scheduling type is in the size of chunks. The size of a chunk is proportional to the number of unassigned iterations divided by the number of the threads. Therefore the size of the chunks decreases.

The minimum size of a chunk is set by `chunk-size`. We determine it in the scheduling clause: `schedule(guided, chunk-size)`. However, the chunk which contains the last iterations may have smaller size than `chunk-size`.

If we do not specify `chunk-size`, it defaults to one.

Here are four examples of the guided scheduling.

```
schedule(guided):
                          *********                          *
              ************                      *******  ***
```

```
                                       *******                   *
****************                             *****       **    *
```

```
schedule(guided, 2):
                ************                      ****     **
                                      *******          ***    **
                             *********
****************                             *****       **    **
```

```
schedule(guided, 4):
                                      *******
                ************                      ****    ****
                             *********
*****************                            *****    ****    ***
```

```
schedule(guided, 8):
                ************                  ********        ***
****************
                                     ********
                             *********                 ********
```

We can see that the size of the chunks is decreasing. First chunk has always 16 iterations. This is because the for loop has 64 iterations and we use 4 threads to parallelize the for loop. If we divide `64 / 4`, we get 16.

We can also see that the minimum chunk size is determined in the schedule clause. The only exception is the last chunk. Its size might be lower then the prescribed minimum size.

The guided scheduling type is appropriate when the iterations are poorly balanced between each other. The initial chunks are larger, because they reduce overhead. The smaller chunks fills the schedule towards the end of the computation and improve load balancing. This scheduling type is especially appropriate when poor load balancing occurs toward the end of the computation.

Auto

The auto scheduling type delegates the decision of the scheduling to the compiler and/or runtime system.

In the following example, the compiler/system determined the static scheduling.

```
schedule(auto):
****************
                ****************
                                ****************
                                                ****************
```

Runtime

The runtime scheduling type defers the decision about the scheduling until the runtime. We already described different ways of specifying the scheduling type in this case. One option is with the environment variable `OMP_SCHEDULE` and the other option is with the function `omp_set_schedule`.

Default

If we do not specify the scheduling type in a for loop

```
#pragma omp parallel for
for (...)
{ ... }
```

OpenMP uses the default scheduling type (defined by the internal control variable `def-sched-var`).

If I do not specify the scheduling type in my machine, I get the following result.

```
default:
****************
                ****************
                                ****************
                                                ****************
```

We recognize that the default scheduling type in my machine is the static scheduling type.

Summary

We learned different ways to specify a scheduling type of a loop. We also described five different scheduling types.

Links:

- Source code of all presented examples

- More information about the scheduling is available in the OpenMP specification on the page 60.

Jaka's Corner OpenMP series:

- OpenMP: Introduction
- OpenMP: For
- OpenMP: Sections
- OpenMP: Monte Carlo method for Pi
- OpenMP: For & Reduction

« Previous post

Next post »