HPC - Exercise 2

Eduardo Gonnelli

02 July 2019

1 Introduction

The objective of this report is to analyze the behavior of different OpenMP schedule clauses and visualize how work is being distributed. The *static* and *dynamic* schedules were applied to a serial program and the a single output was printed employing a *print_usage()* given function. The workload distribution schedule for parallel loops was analyzed for different number of chunks. The code was written in C++ programming language and the schedules were implemented on a single parallel region. The code was executed on Ulysses Cluster and the number of threads was equal 10.

2 Schedule Clause

The schedule clause is used to control the manner in which loop iterations are distributed over threads. It may be used to explicitly specify the mapping of loop iterations onto threads. This procedure might have a major impact on the performance of a program. By default, if the schedule clause is not specified, the choice is made by the compiler and is therefore system-dependent. However, the OpenMP also provides the user to prescribe the way in which the loop iterations should be distributed. Regarding the schedule clause application, it is supported only on the loop construct.

There are three different scheduling types: static, dynamic, and guided. All three take an optional chunk parameter to control how many iterations are to be processed each time a thread gets assigned new work. The following subsections will brief describe some characteristics of *static* and *dynamic* schedules. The scope of this exercise is to analyze the behavior of static and dynamic schedules. Due to this fact, the *guided* schedule is not included.

2.1 Static Schedule

In the static schedule, the iterations are divided into chunks of size *chunk_size*. The chunks are assigned to the threads statically in a round-robin manner, in the order of the thread number. The last chunk to be assigned may have a smaller number of iterations. When no chunk size is specified, the iteration space is divided into chunks that are approximately equal in size. Each thread roughly gets assigned the same amount of work. Normally, the static schedule is set as default on many OpenMP compilers, to be used in the absence of an explicit schedule clause.

2.2 Dynamic Schedule

In the *dynamic* schedule the iterations are assigned to threads as the threads request them. The thread executes the chunk of iterations (controlled through the chunk size parameter), then requests another chunk until there are no more chunks to work on. The last chunk may have fewer iterations than chunk size. It is more suitable in the case of a load imbalance. In some compilers, when no chunk size is specified, it defaults to 1.

3 Results

In this section the results related to the schedule clauses will be presented. The node cn08-17 were used to execute the code. To specify the number of threads the environment variable *export OMP_NUM_THREADS=10* was called. The size of the problem (N) was defined as 110. The initial number was 250, but for visualization purposes, it was readjusted without loss of generality. The Figure 1 shows the region of the parallel code containing the schedule clause *schedule(static,10)*.

```
//static, with chunk size 10
#pragma omp for schedule(static,10)
     for(int i = 0; i < N; ++i)
        {
          a[i] = omp_get_thread_num();
        }
//Only one Thread will print the result
#pragma omp single
{    std::cout << "This is the Schedule(static,10)" << "\n";
     std::cout << "\n";
     print_usage(a, N, nthreads);
     std::cout << "\n";
}
```

Figure 1: Piece of code written in C++ employing the schedule(static,10)

The Table 1 contains the values for schedules configuration for each parallel loop.

Table 1: Schedules Configuration.

Static Schedule	Dynamic Schedule
schedule(static)	schedule(dynamic)
schedule(static,1)	schedule(dynamic,1)
schedule(static,10)	schedule(dynamic,10)

The mapping of iterations onto ten threads for static and dynamic scheduling algorithms for a loop of length N = 110 is shown. Clearly, the dynamic policy gives rise to much more dynamic workload allocation scheme. It can be seen on Figures 2 and 3 that the behavior is different between the *schedule(static)* and *schedule(static,1)*. The *schedule(static)* divided the amount of work equally, i.e., N / 10 (Problem size / Total number of threads). Therefore, each thread was responsible for printing 11 asterisks. On the other hand, the Figure 4 shows the behavior when setting the chunk_size = 10; the amount of work is divided in parts of 10 asterisks.

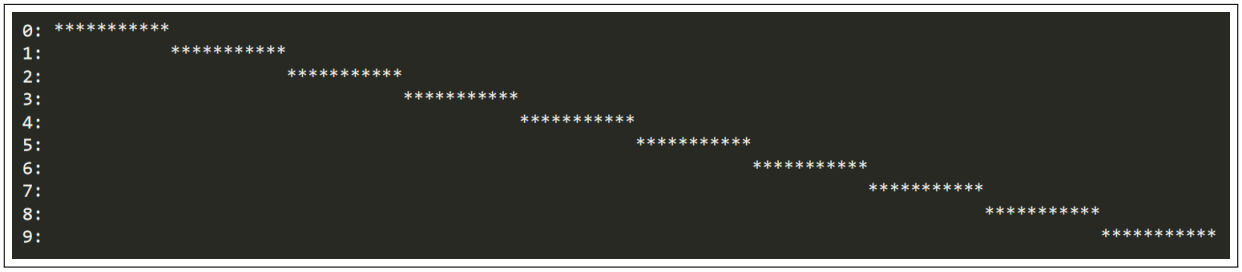

Figure 2: Static Schedule, without chunk size specification

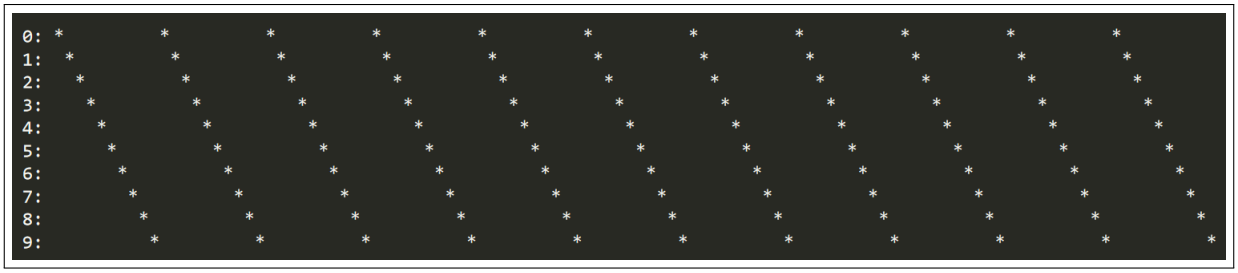

Figure 3: Static Schedule, chunk_size = 1

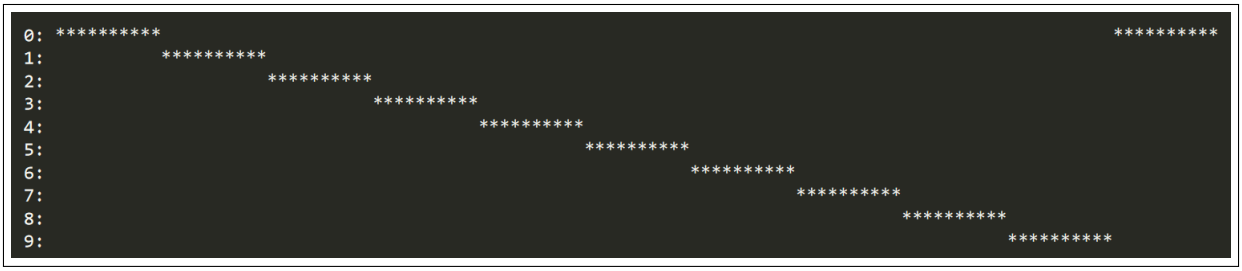

Figure 4: Static Schedule, chunk_size = 10

Analyzing the dynamic results on Figures 5, 6, and 7, one can be noticed that there is no criterion by which thread will be initialized first. It can be seen that the work began on thread number 8, 2 and 5, for *schedule(dynamic)*, *schedule(dynamic,1)* and *schedule(dynamic,10)*, respectively. The following figures show the output for dynamic schedules considered in this work.

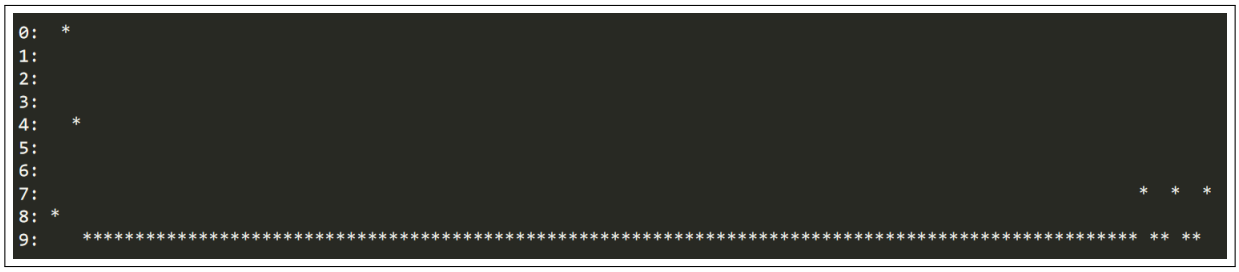

Figure 5: Dynamic Schedule, without chunk size specification

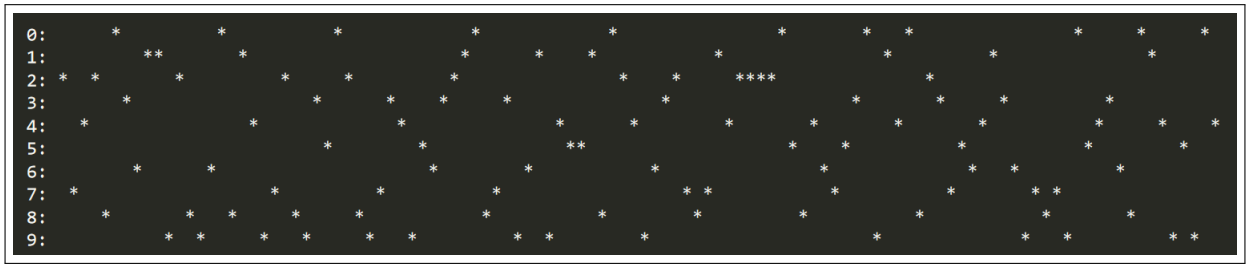

Figure 6: Dynamic Schedule, chunk_size = 1

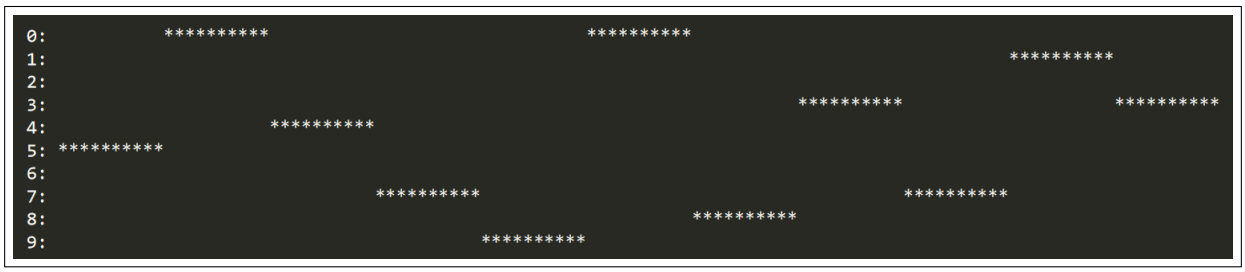

Figure 7: Dynamic Schedule, chunk_size = 10

References

[1] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. *Using OpenMP-Portable Shared Memory Parallel Programming*. The MIT Press, Cambridge, 2008.

[2] Ruud van der Pas, Eric Stotzer, and Christian Terboven. *Using OpenMP-The Next Step: Affinity, Accelerators, Tasking, and SIMD*. The MIT Press, Cambridge, 2017.

Appendices

This section contains the codes developed for this work. The compile command for OpenMP was *g++ -fopenmp schedule.cc -o schedule.o*.

A Job Script

Listing 1: Job Script

```
#!/bin/bash
#PBS -l nodes=1:ppn=20
#PBS -l walltime=02:30:00
#PBS -q regular

cd /home/egonnell/igirotto/ex02

#load the modules

module load gnu/4.9.2
#call 10 threads, run the program and salve on txt file
export OMP_NUM_THREADS=10; ./schedule.o > 110N.txt
```

B C++ code

Listing 2: Serial Code

```
#include <iostream>

void print_usage( int * a, int N, int nthreads );

int main(int argc, char const *argv[])
{
   const int N = 110;
   int a[N];
   int thread_id = 0;
   int nthreads = 1;

    for(int i = 0; i < N; ++i){
      a[i] = thread_id;
    }

print_usage(a, N, nthreads);
return 0;
}

void print_usage( int * a, int N, int nthreads ) {

  int tid, i;
  for( tid = 0; tid < nthreads; ++tid ) {

      std::cout << tid << ": ";

    for( i = 0; i < N; ++i ) {

      if( a[ i ] == tid) std::cout << "*";
      else std::cout << " ";
    }
    std::cout << std::endl;
  }
}
```

The following code contains one parallel region with *schedule(static)*, *schedule(static,1)*, *schedule(static,10)*, *schedule(dynamic)*, *schedule(dynamic,1)*, and *schedule(dynamic,10)*. The code was written in C++ language.

Listing 3: Schedule Code with one parallel region

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

void print_usage( int * a, int N, int nthreads );

int main(int argc, char const *argv[])
{
   const int N =80;
   int a[N];
   int thread_id = 0;
   int nthreads = 1;

    for(int i = 0; i < N; ++i){
      a[i] = thread_id;
    }

print_usage(a, N, nthreads);
std::cout << "\n";
```

```
#pragma omp parallel
{
nthreads = omp_get_num_threads();
#pragma omp for schedule(static)
     for(int i = 0; i < N; ++i)
       {
         a[i] = omp_get_thread_num();
       }
//Only one Thread will print the result
#pragma omp single
{
    std::cout << "This is the Schedule(static)" << "\n";
    std::cout << "\n";
    print_usage(a, N, nthreads);
    std::cout << "\n";
}

//static, with chunk size 1
#pragma omp for schedule(static,1)
     for(int i = 0; i < N; ++i)
       {
         a[i] = omp_get_thread_num();
       }
//Only one Thread will print the result
#pragma omp single
{
    std::cout << "This is the Schedule(static,1)" << "\n";
    std::cout << "\n";
    print_usage(a, N, nthreads);
    std::cout << "\n";
}


//static, with chunk size 10
#pragma omp for schedule(static,10)
     for(int i = 0; i < N; ++i)
       {
         a[i] = omp_get_thread_num();
       }
//Only one Thread will print the result
#pragma omp single
{
    std::cout << "This is the Schedule(static,10)" << "\n";
    std::cout << "\n";
    print_usage(a, N, nthreads);
    std::cout << "\n";
}

//dynamic
#pragma omp for schedule(dynamic)
     for(int i = 0; i < N; ++i)
       {
         a[i] = omp_get_thread_num();
       }
//Only one Thread will print the result
#pragma omp single
{
    std::cout << "This is the Schedule(dynamic)" << "\n";
    std::cout << "\n";
    print_usage(a, N, nthreads);
    std::cout << "\n";
```

```
}

//dynamic, with chunk size 1
#pragma omp for schedule(dynamic,1)
     for(int i = 0; i < N; ++i)
       {
         a[i] = omp_get_thread_num();
       }
//Only one Thread will print the result
#pragma omp single
{
    std::cout << "This is the Schedule(dynamic,1)" << "\n";
    std::cout << "\n";
    print_usage(a, N, nthreads);
    std::cout << "\n";
}

//dynamic, with chunk size 10
#pragma omp for schedule(dynamic,10)
     for(int i = 0; i < N; ++i)
       {
         a[i] = omp_get_thread_num();
       }
//Only one Thread will print the result
#pragma omp single
{
    std::cout << "This is the Schedule(dynamic,10)" << "\n";
    std::cout << "\n";
    print_usage(a, N, nthreads);
    std::cout << "\n";
}

 }


return 0;
}

void print_usage( int * a, int N, int nthreads ) {
  int tid, i;
  for( tid = 0; tid < nthreads; ++tid ) {

      std::cout << tid << ": ";

    for( i = 0; i < N; ++i ) {

      if( a[ i ] == tid) std::cout << "*";
      else std::cout << " ";
    }
    std::cout << std::endl;
  }
}
```