Fifth deliverable

Álvaro Martínez Arroyo

Daniel García Romero

par2303

Fall 2015-2016

INDEX

1. Analysis with Tareador

1. Include the relevant parts of the modified solver-tareador.c code and comment where the calls to the Tareador API have been placed. Comment also about the task graph generated and the causes of the dependences that appear in the two solvers: *Jacobi* and *Gauss-Seidel*. How will you protect them in the parallel OpenMP code?

If we add *tareador_start_task* and *tareador_end_task* at the beginning and at the end, respectively, of the Jacobi's internal loop, we obtain the task graph of the Figure 1.

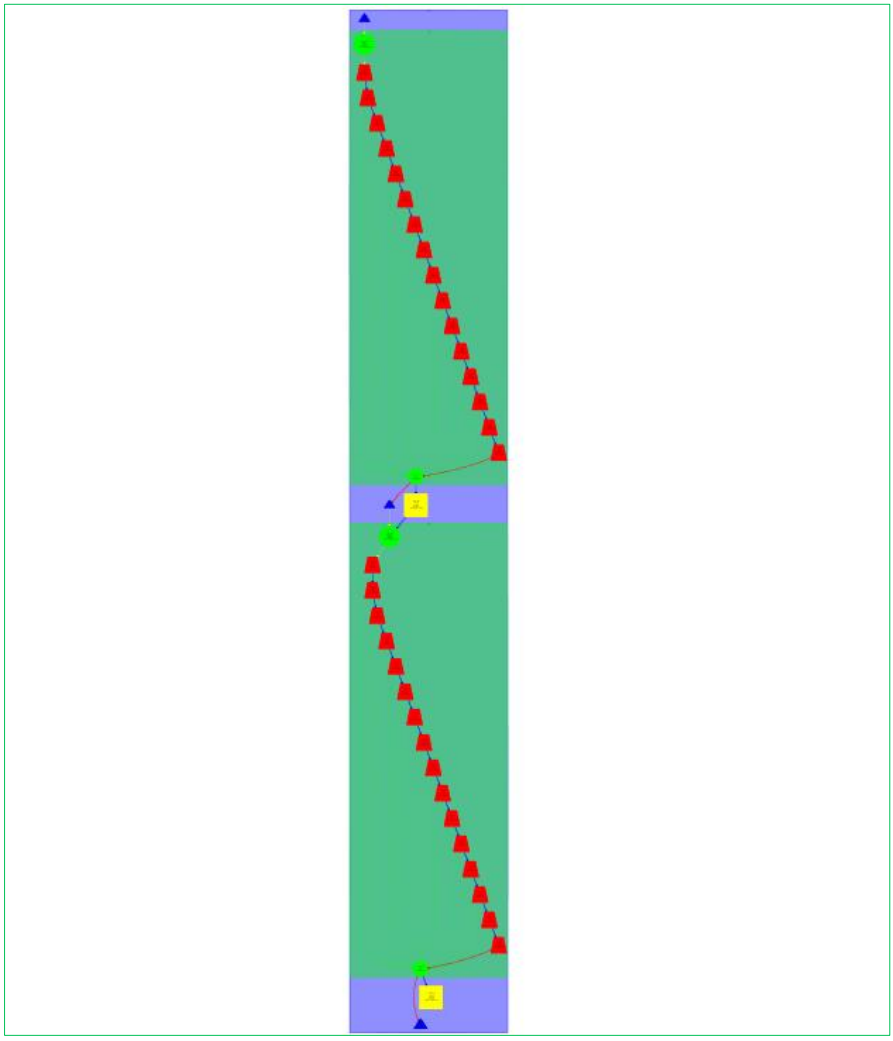

Figure 1. Dependences of Jacobi's task graph

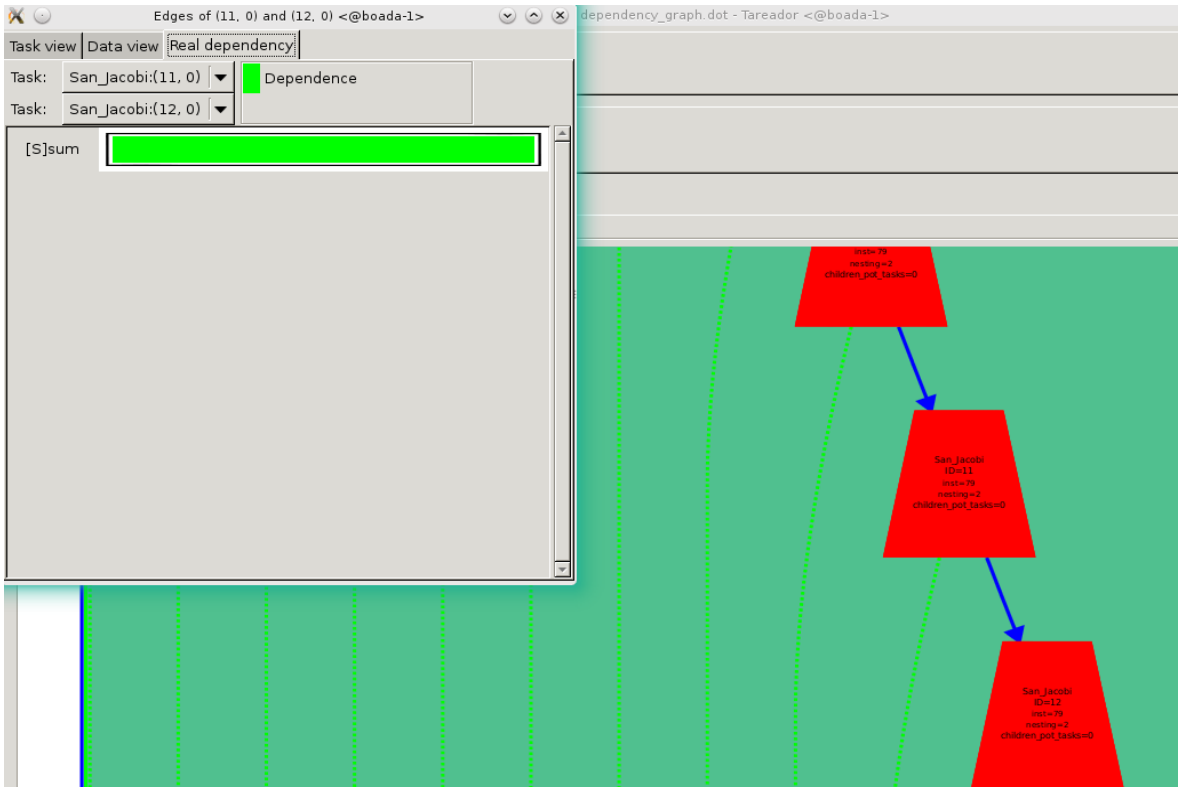

Figure 2. Dependences of Jacobi's task graph using Dataview option in Tareador

In order to find out what is generating these dependences, we right-click any dependence and select *Dataview* and *Edge*. Then we can see that the variable that causes the serialization of tasks is *sum* (Figure 2).

If we add *tareador_disable_object(&sum)* after *tareador_start_task* and *tareador_enable_object(&sum)* before *tareador_end_task*, we obtain the task graph of the Figure 3.

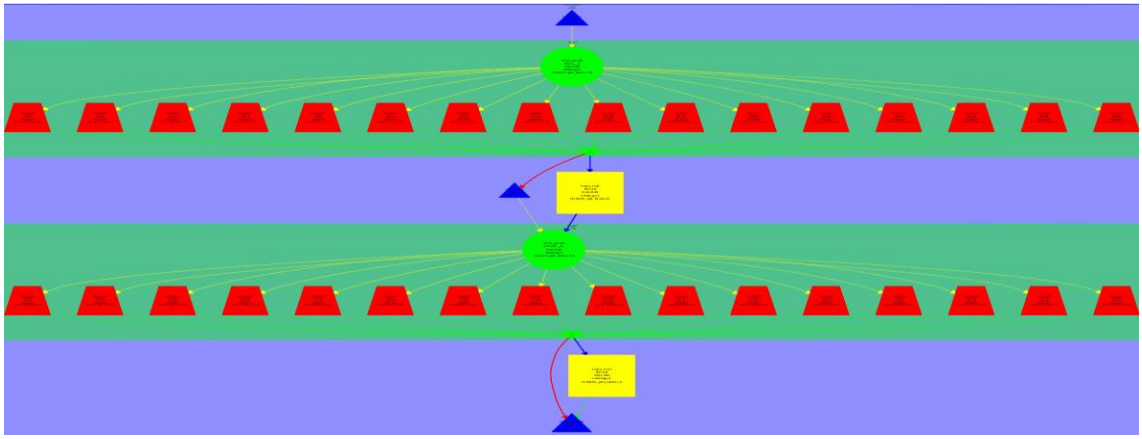

Figure 3. Jacobi's task graph without the dependences of the sum variable

As we have done with the Jacobi solver, we add *tareador_start_task* and *tareador_end_task* at the beginning and at the end, respectively, of the Gauss-Seidel's internal loop to obtain the task graph of the Figure 4.

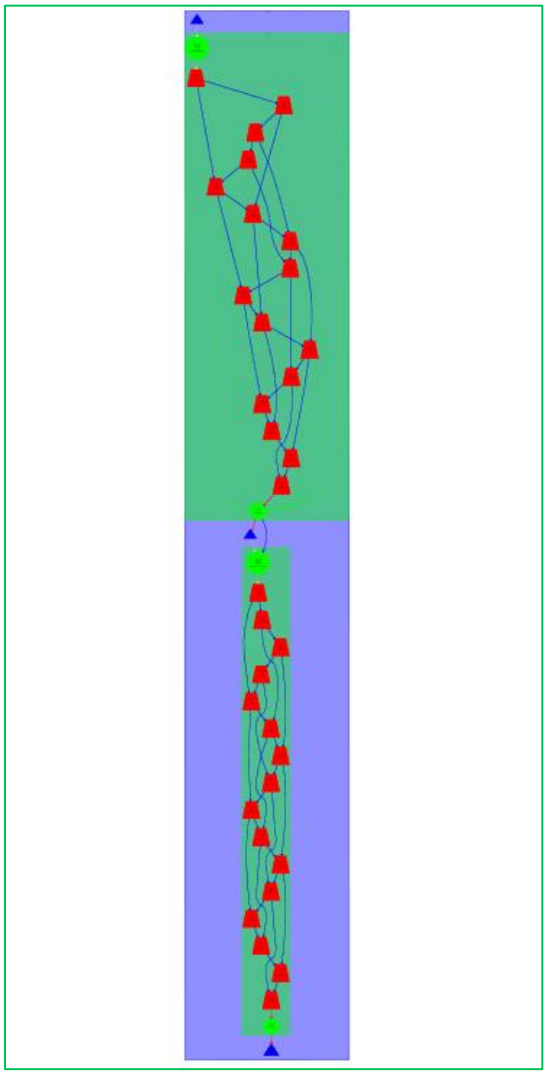

Figure 4. Dependences of Gauss-Seidel's task graph

In order to find out what is generating these dependences, we right-click any dependence and select *Dataview* and *Edge*. Then we can see that the *sum* variable and another dependence cause the serialization of tasks (Figure 5). This dependence can't be disabled and is generated because the *u* matrix is read and written, whereas in Jacobi's code, the *u* matrix is only read and a temporal matrix called *utmp* is used.

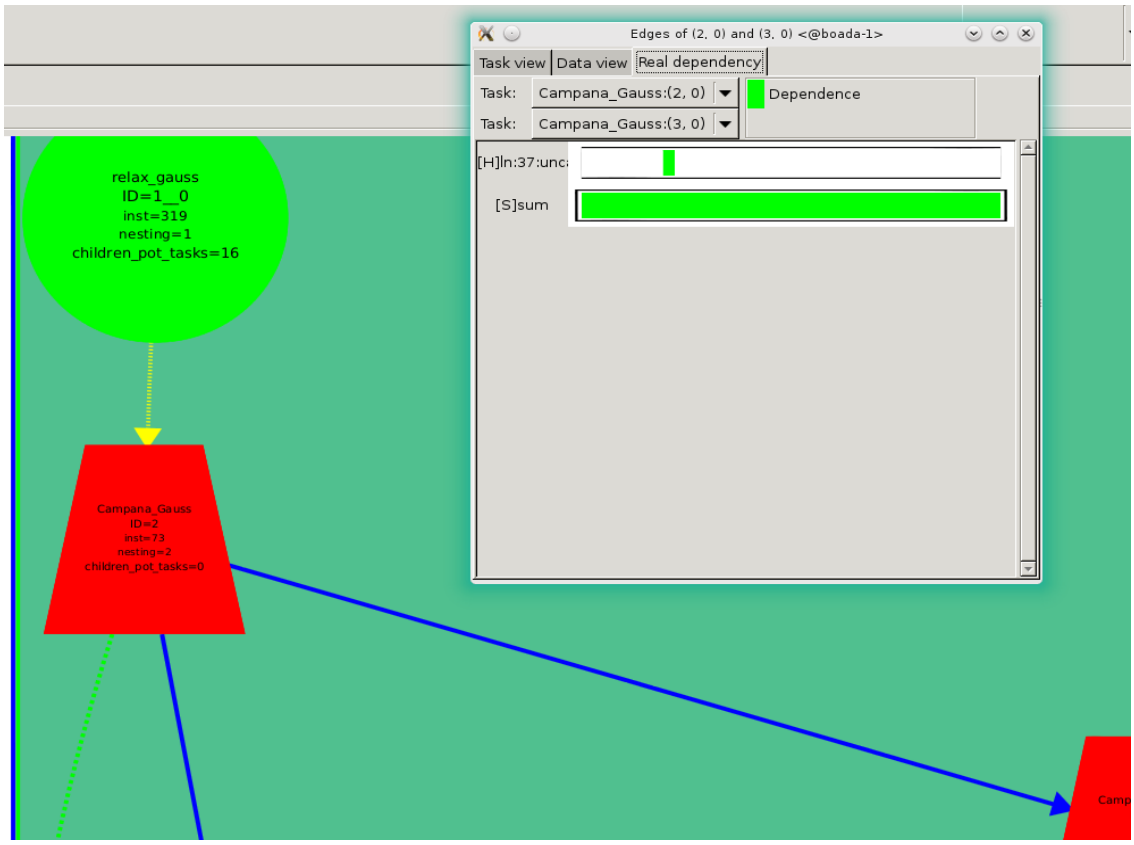

Figure 5. Dependences of Gauss-Seidel's using Dataview option in Tareador

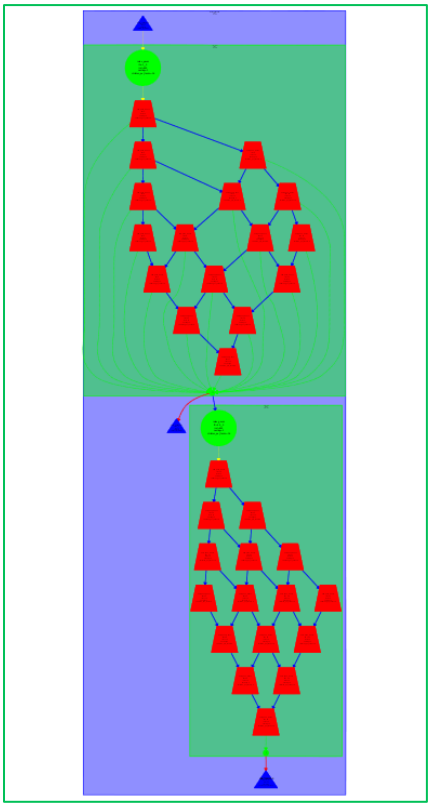

Figure 6. Gauss-Seidel's task graph without the dependences of the sum variable

If we add *tareador_disable_object(&sum)* after *tareador_start_task* and *tareador_enable_object(&sum)* before *tareador_end_task*, we obtain the task graph of the Figure 6.

If we compare the task graphs with and without the dependences of the *sum* variable, we can observe that we are increasing the parallelism because more tasks can be done in parallel.

In order to protect the *sum* variable dependence in an OpenMP parallelization based on *#pragma omp for*, we can use a *reduction(+ : sum)*, so each thread will have a private *sum* variable where storing the increase and when all of them have finished their assigned iterations, the sum of the private *sum* variables will be stored in the *sum* variable.

In case of using a parallelization strategy based on *#pragma omp task* for the Gauss-Seidel solver, we can use the *depend* clause for the *u* matrix dependences.

2. OpenMP parallelization and execution analysis: Jacobi

1. Describe the data decomposition strategy that is applied to solve the problem, including a picture with the part of the data structure that is assigned to each processor.

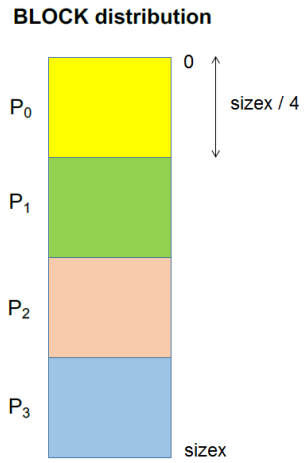

Figure 7. Data decomposition for Jacobi solver

We use a Geometric Block Data Decomposition, where the block which is assigned to each thread is a submatrix of (sizex/howmany)*sizey elements. The mapping of these blocks starts with the left upper submatrix, from left to right and from top to bottom. For each submatrix we calculate its i_start and i_end depending on the blockid (thread).

2. Include the relevant portions of the parallel code that you implemented to solve the heat equation using the Jacobi solver, commenting whatever necessary. Including captures of Paraver windows to justify your explanations and the differences observed in the execution.

We add *#pragma omp parallel for private (diff) reduction (+:sum)* in relax_jacobi because we want that:

- Each thread has a private *sum* variable where storing the increase, and when all of them have finished their assigned iterations, the sum of the private *sum* variables will be stored in the *sum* variable.

- Each thread has a local copy of the *diff* variable.

In addition, we put *#pragma omp parallel for* in copy_mat in order to improve the parallelism.

3. Include the speed-up (strong scalability) plots that have been obtained for the different numbers of processors. Reason about the performance that is observed.

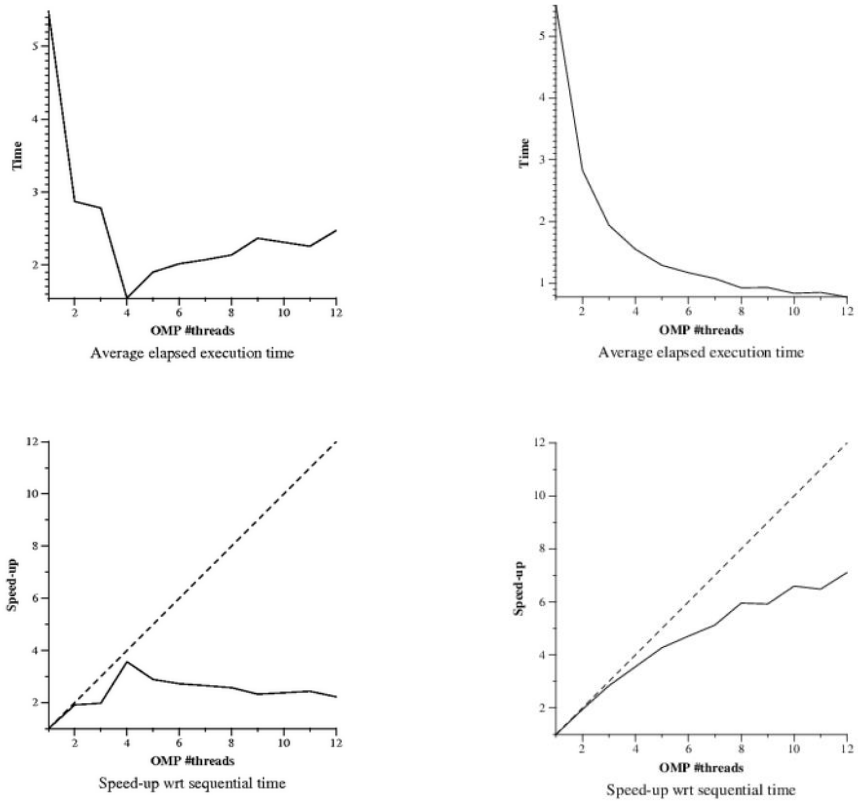

Figure 8. Speed-up plots obtained for Jacobi solver

As we can see in the Figure 8, if we parallelize copy_mat, the execution time decreases and the speed-up is better.

3. OpenMP parallelization and execution analysis: Gauss-Seidel

1. Include the relevant portions of the parallel code that implements the Gauss-Seidel solver, commenting how you implemented the synchronization between threads.

We add *#pragma omp parallel for private (unew, diff) reduction (+:sum)* in relax_gauss because we want that:

- Each thread has a private *sum* variable where storing the increase, and when all of them have finished their assigned iterations, the sum of the private *sum* variables will be stored in the *sum* variable.

- Each thread has a local copy of the *diff* and *unew* variables.

In the calculation of the *unew* value, we find dependences with the top and the left elements (*wavefront*), so we need a blocking technique.

The *blockfinished* vector includes the last columns each thread has done. When a thread computes a block, increments in one its blockfinished cell.

The while loop ensures that the threads will not calculate the *unew* value until the top block has been computed (the left block doesn't matter because of the way that the threads calculate the blocks, which is from the left to the right).

In addition*, #pragma omp flush* is used to enforce memory consistency.

2. Include the speed-up (strong scalability) plot that has been obtained for the different numbers of processors. Reason about the performance that is observed, including captures of Paraver windows to justify your explanations.

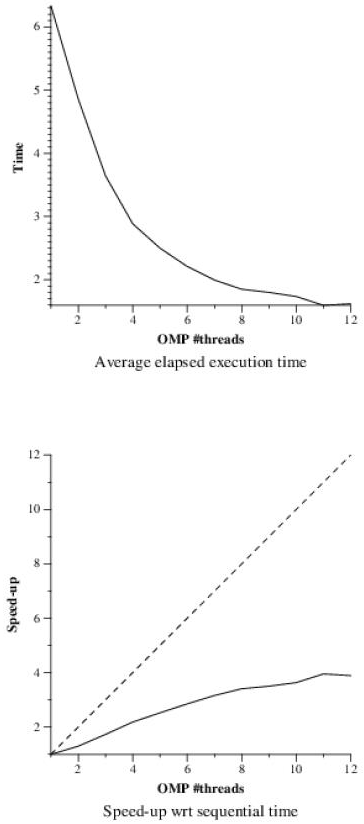

Figure 9. Speed-up plot obtained for Gauss-Seidel solver

If we compare the figures 8 and 9, we can see that the speedup of Gauss-Seidel solver is worse than Jacobi solver because of the matrix dependences.

3. Explain how did you obtain the optimum value for the ratio computation/synchronization in the parallelization of this solver for 8 threads.

-