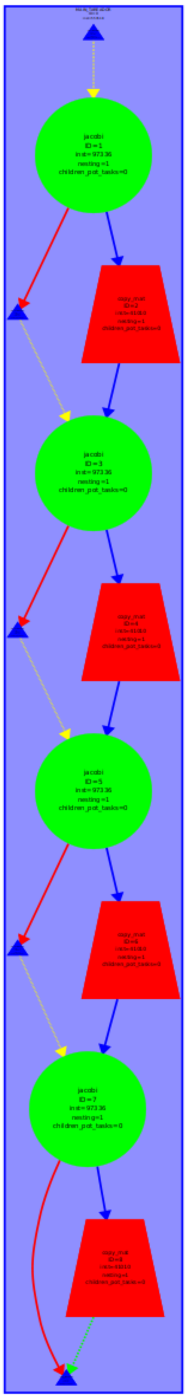
ACO Laboratory Assignment
Lab 5: Geometric (data) decomposition using implicit tasks: heat diffusion equation

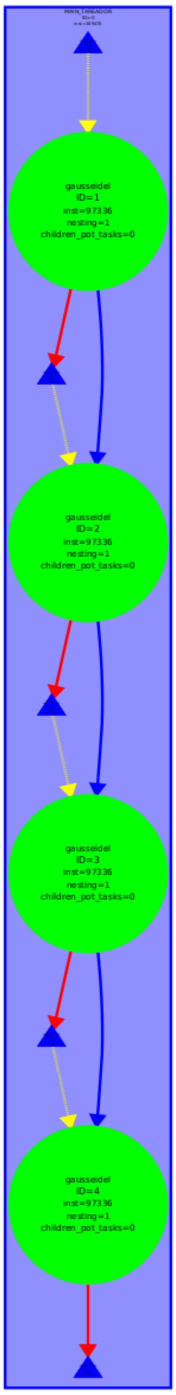
Oriol Amate Sabat i Adrian Garcia Campillo

Index

2.Sequential heat diffusion program and analysis with Tareador

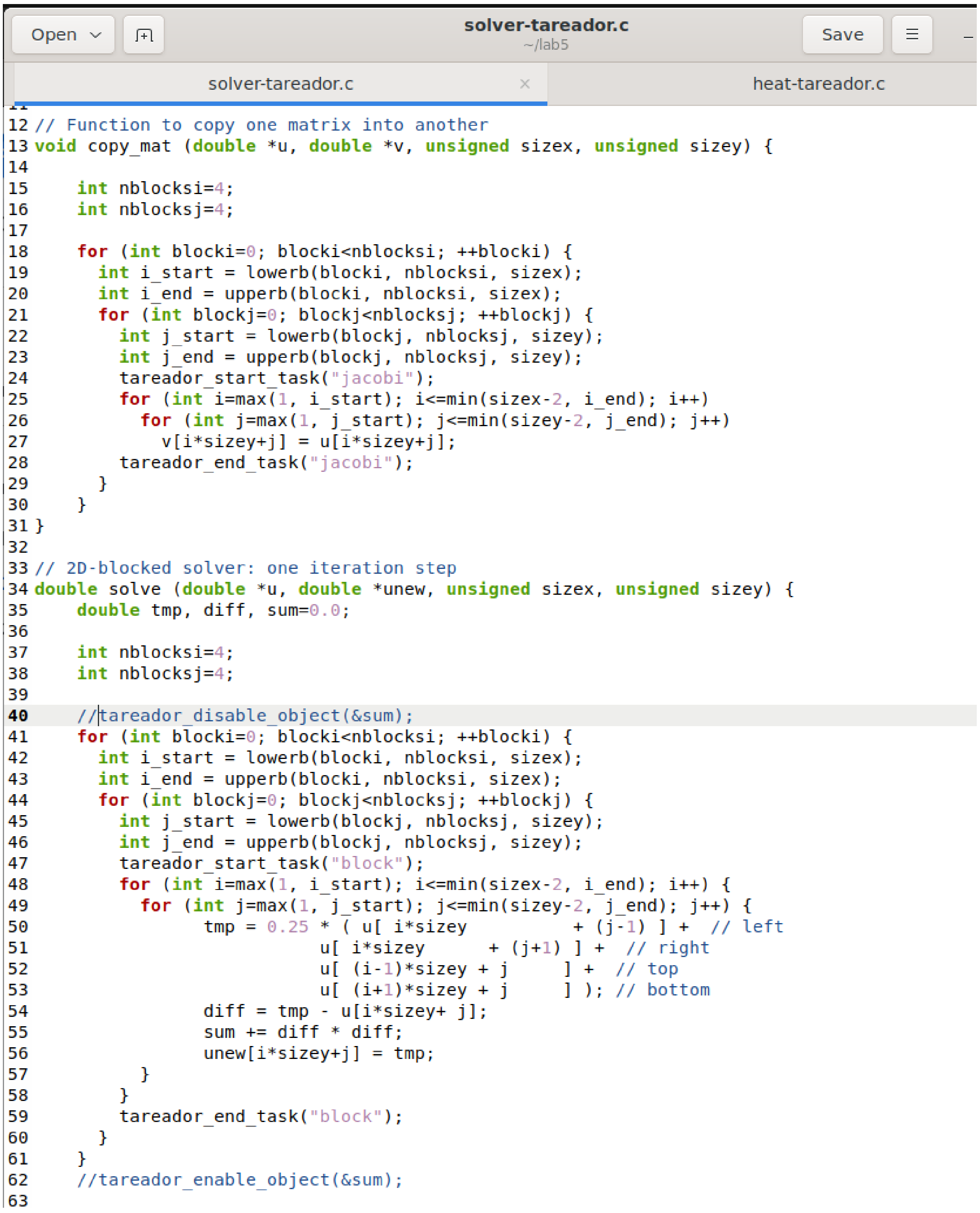
Include the task dependency graph shown by Tareador. Is there any parallelism that can be exploited at this granularity level?





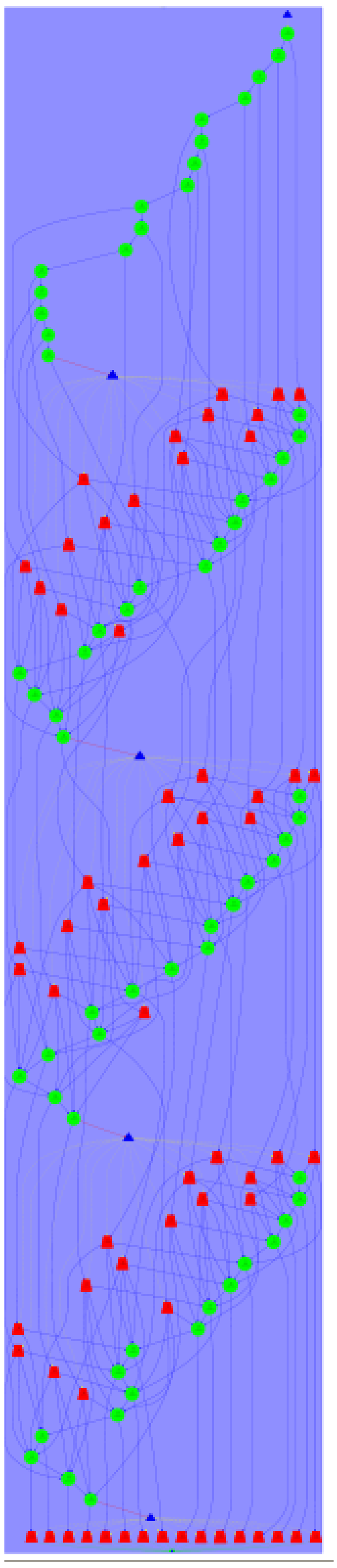
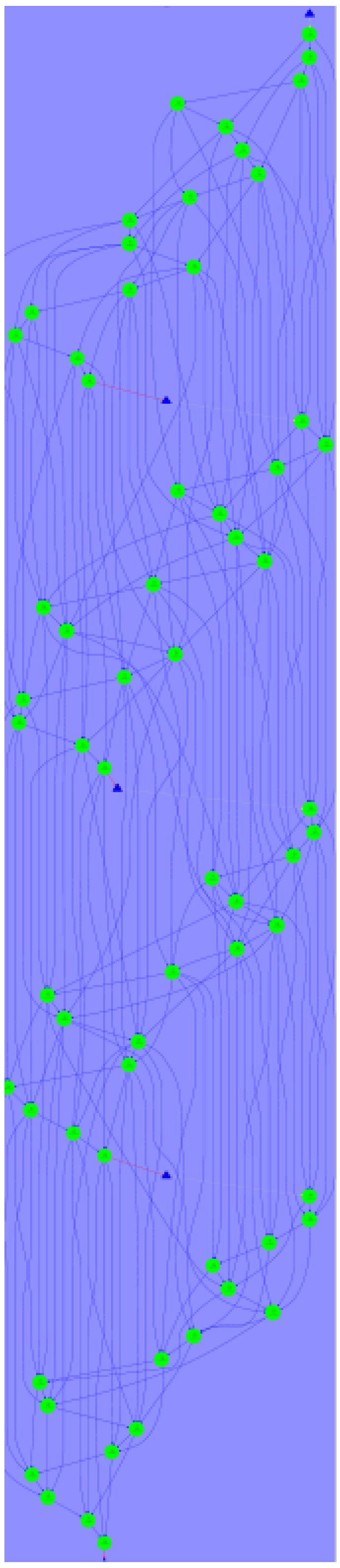
Podem paral·lelitzar copy_mat y solve.

Include the excerpt of the code that you have modified in order to specify one task per block.

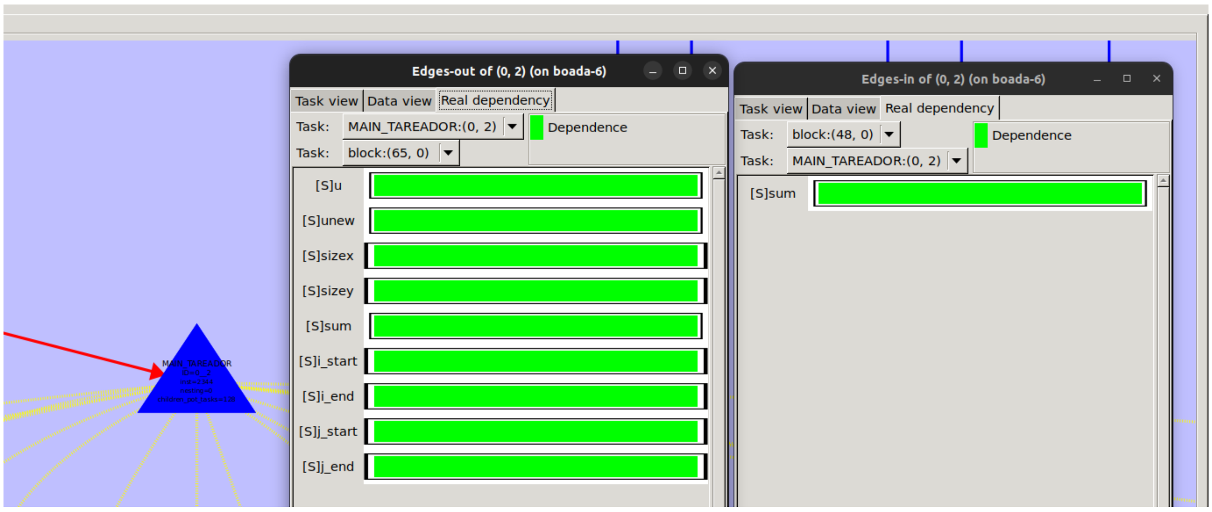


```c
// Function to copy one matrix into another
void copy_mat (double *u, double *v, unsigned sizex, unsigned sizey) {

    int nblocksi=4;
    int nblocksj=4;

    for (int blocki=0; blocki<nblocksi; ++blocki) {
      int i_start = lowerb(blocki, nblocksi, sizex);
      int i_end = upperb(blocki, nblocksi, sizex);
      for (int blockj=0; blockj<nblocksj; ++blockj) {
        int j_start = lowerb(blockj, nblocksj, sizey);
        int j_end = upperb(blockj, nblocksj, sizey);
        tareador_start_task("jacobi");
        for (int i=max(1, i_start); i<=min(sizex-2, i_end); i++)
          for (int j=max(1, j_start); j<=min(sizey-2, j_end); j++)
            v[i*sizey+j] = u[i*sizey+j];
        tareador_end_task("jacobi");
      }
    }
}

// 2D-blocked solver: one iteration step
double solve (double *u, double *unew, unsigned sizex, unsigned sizey) {
    double tmp, diff, sum=0.0;

    int nblocksi=4;
    int nblocksj=4;

    //tareador_disable_object(&sum);
    for (int blocki=0; blocki<nblocksi; ++blocki) {
      int i_start = lowerb(blocki, nblocksi, sizex);
      int i_end = upperb(blocki, nblocksi, sizex);
      for (int blockj=0; blockj<nblocksj; ++blockj) {
        int j_start = lowerb(blockj, nblocksj, sizey);
        int j_end = upperb(blockj, nblocksj, sizey);
        tareador_start_task("block");
        for (int i=max(1, i_start); i<=min(sizex-2, i_end); i++) {
          for (int j=max(1, j_start); j<=min(sizey-2, j_end); j++) {
                tmp = 0.25 * ( u[ i*sizey          + (j-1) ] +  // left
                               u[ i*sizey      + (j+1) ] +  // right
                               u[ (i-1)*sizey + j      ] +  // top
                               u[ (i+1)*sizey + j      ] ); // bottom
                diff = tmp - u[i*sizey+ j];
                sum += diff * diff;
                unew[i*sizey+j] = tmp;
          }
        }
        tareador_end_task("block");
      }
    }
    //tareador_enable_object(&sum);

```

Jacobi	Gauss

Which variable is causing the serialisation of all the tasks? Use the Dataview option in Tareador to identify it.
Jacobi



Edges-out of (0, 2): Task: MAIN_TAREADOR:(0, 2) → Task: block:(65, 0)

🟩 Dependence

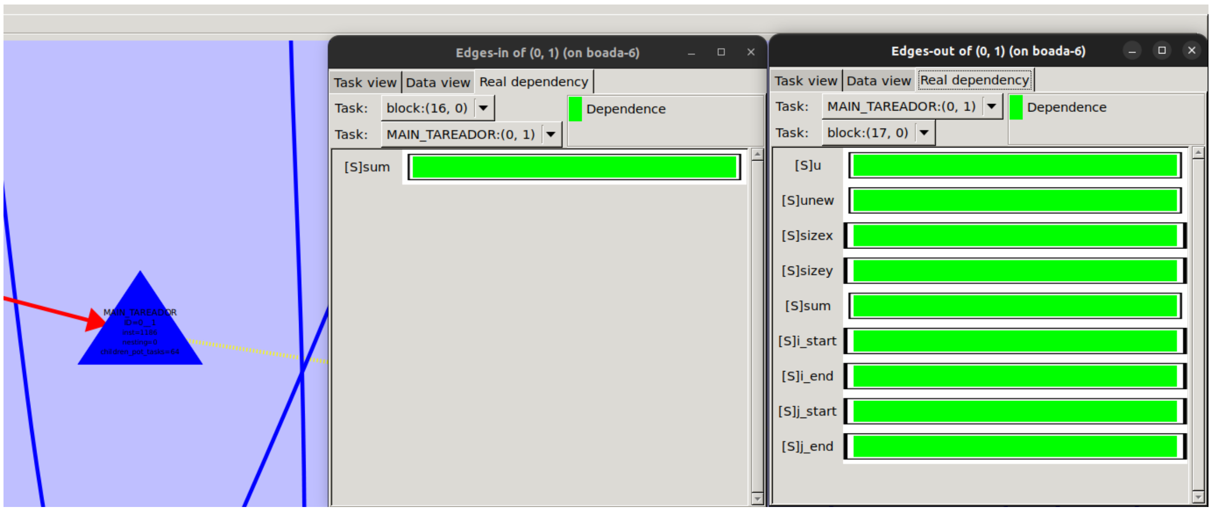
[S]u	🟩
[S]unew	🟩
[S]sizex	🟩
[S]sizey	🟩
[S]sum	🟩
[S]i_start	🟩
[S]i_end	🟩
[S]j_start	🟩
[S]j_end	🟩

Edges-in of (0, 2): Task: block:(48, 0) → Task: MAIN_TAREADOR:(0, 2)

🟩 Dependence

[S]sum	🟩

Gauss



Edges-in of (0, 1): Task: block:(16, 0) → Task: MAIN_TAREADOR:(0, 1)

🟩 Dependence

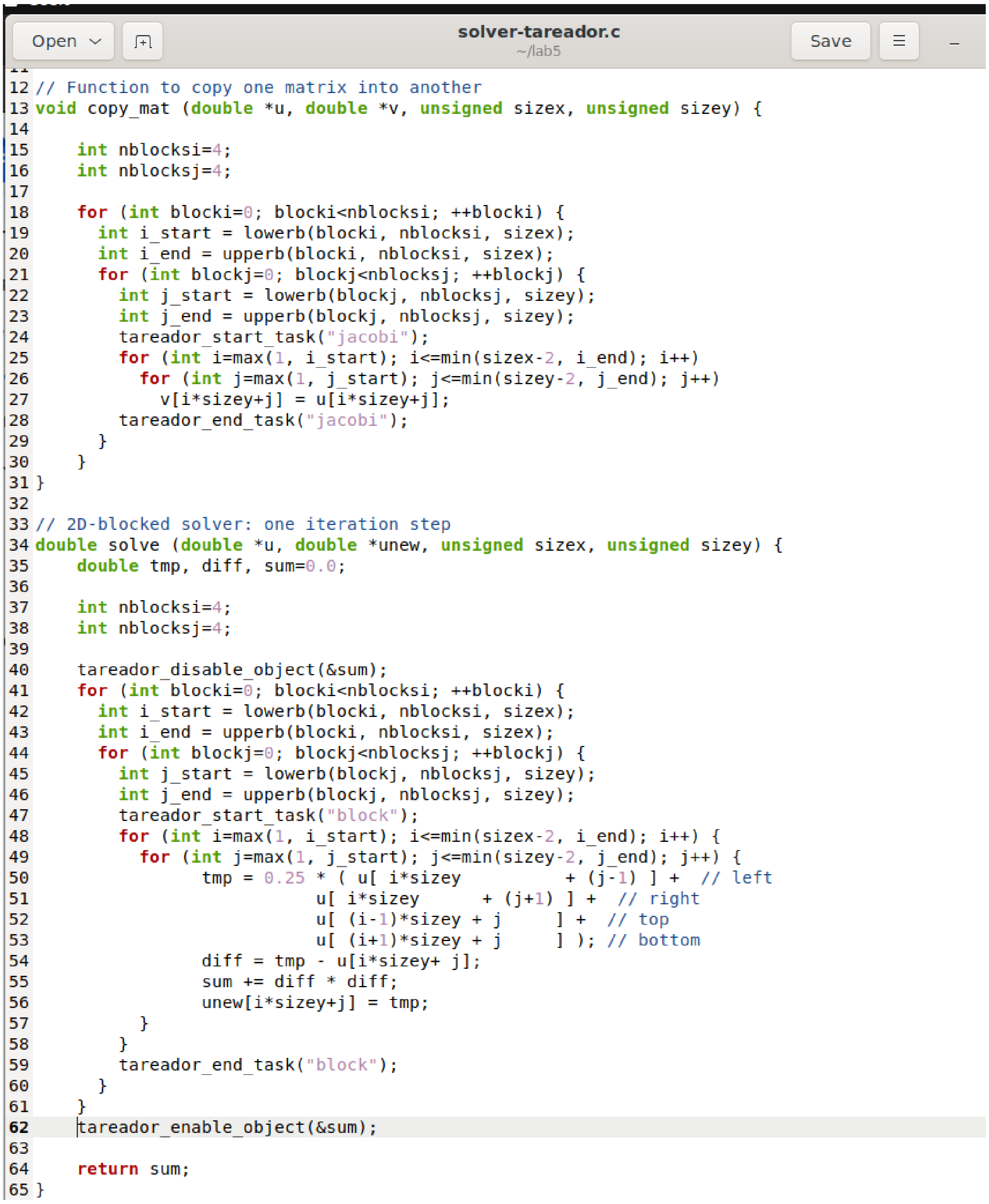
[S]sum	🟩

Edges-out of (0, 1): Task: MAIN_TAREADOR:(0, 1) → Task: block:(17, 0)

🟩 Dependence

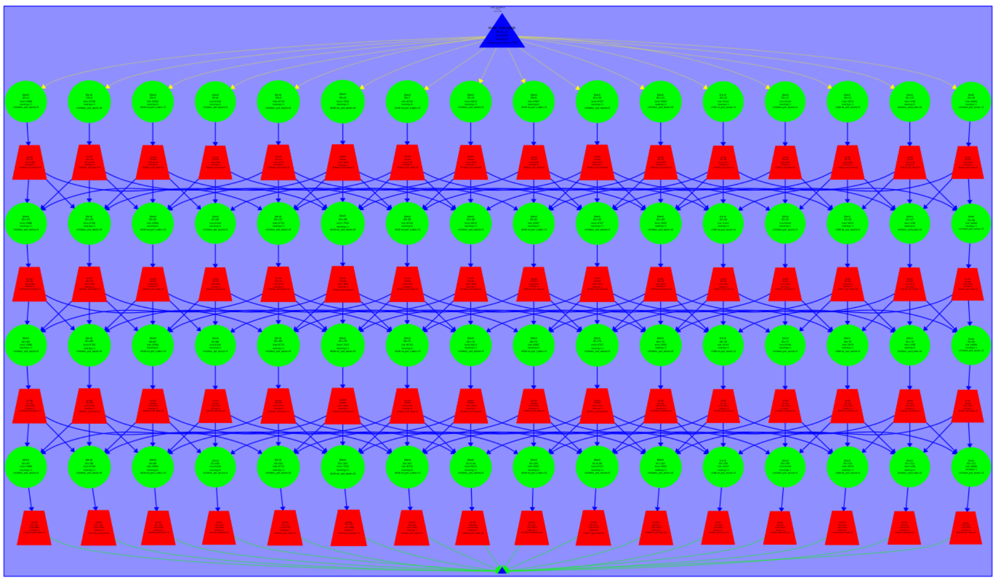
[S]u	🟩
[S]unew	🟩
[S]sizex	🟩
[S]sizey	🟩
[S]sum	🟩
[S]i_start	🟩
[S]i_end	🟩
[S]j_start	🟩
[S]j_end	🟩

In order to emulate the effect of protecting the dependences caused by this variable, you can use the tareador disable object and tareador enable object calls, already introduced in the code as comments. With these calls you are telling to Tareador to filter the dependences caused by the variable indicated as object. Uncomment them, recompile and execute.

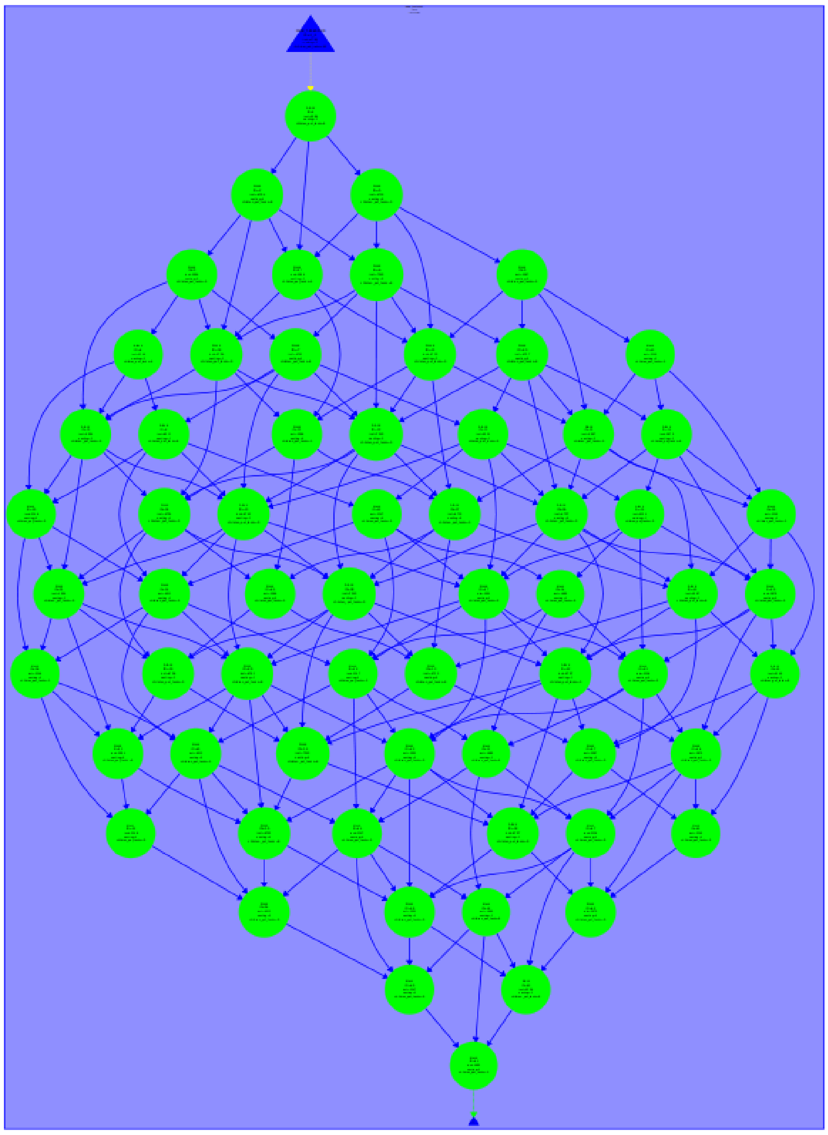


```c
// Function to copy one matrix into another
void copy_mat (double *u, double *v, unsigned sizex, unsigned sizey) {

    int nblocksi=4;
    int nblocksj=4;

    for (int blocki=0; blocki<nblocksi; ++blocki) {
      int i_start = lowerb(blocki, nblocksi, sizex);
      int i_end = upperb(blocki, nblocksi, sizex);
      for (int blockj=0; blockj<nblocksj; ++blockj) {
        int j_start = lowerb(blockj, nblocksj, sizey);
        int j_end = upperb(blockj, nblocksj, sizey);
        tareador_start_task("jacobi");
        for (int i=max(1, i_start); i<=min(sizex-2, i_end); i++)
          for (int j=max(1, j_start); j<=min(sizey-2, j_end); j++)
            v[i*sizey+j] = u[i*sizey+j];
        tareador_end_task("jacobi");
      }
    }
}

// 2D-blocked solver: one iteration step
double solve (double *u, double *unew, unsigned sizex, unsigned sizey) {
    double tmp, diff, sum=0.0;

    int nblocksi=4;
    int nblocksj=4;

    tareador_disable_object(&sum);
    for (int blocki=0; blocki<nblocksi; ++blocki) {
      int i_start = lowerb(blocki, nblocksi, sizex);
      int i_end = upperb(blocki, nblocksi, sizex);
      for (int blockj=0; blockj<nblocksj; ++blockj) {
        int j_start = lowerb(blockj, nblocksj, sizey);
        int j_end = upperb(blockj, nblocksj, sizey);
        tareador_start_task("block");
        for (int i=max(1, i_start); i<=min(sizex-2, i_end); i++) {
          for (int j=max(1, j_start); j<=min(sizey-2, j_end); j++) {
                tmp = 0.25 * ( u[ i*sizey           + (j-1) ] +  // left
                           u[ i*sizey       + (j+1) ] +  // right
                           u[ (i-1)*sizey + j      ] +  // top
                           u[ (i+1)*sizey + j      ] ); // bottom
                diff = tmp - u[i*sizey+ j];
                sum += diff * diff;
                unew[i*sizey+j] = tmp;
          }
        }
        tareador_end_task("block");
      }
    }
    tareador_enable_object(&sum);

    return sum;
}
```

JACOBI

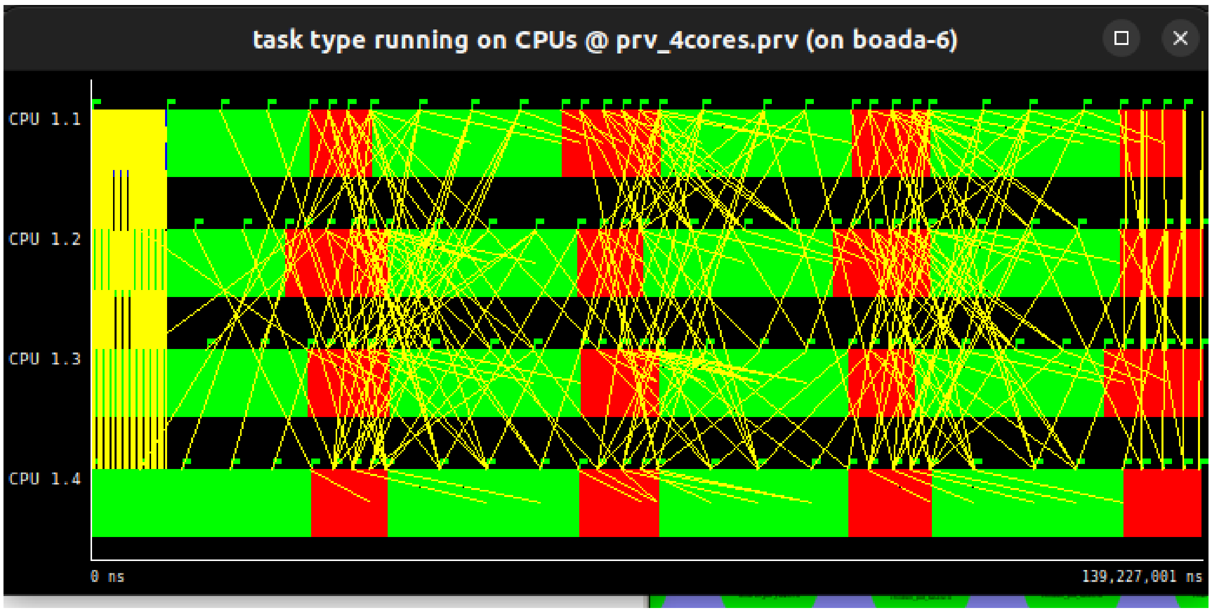


GAUSS

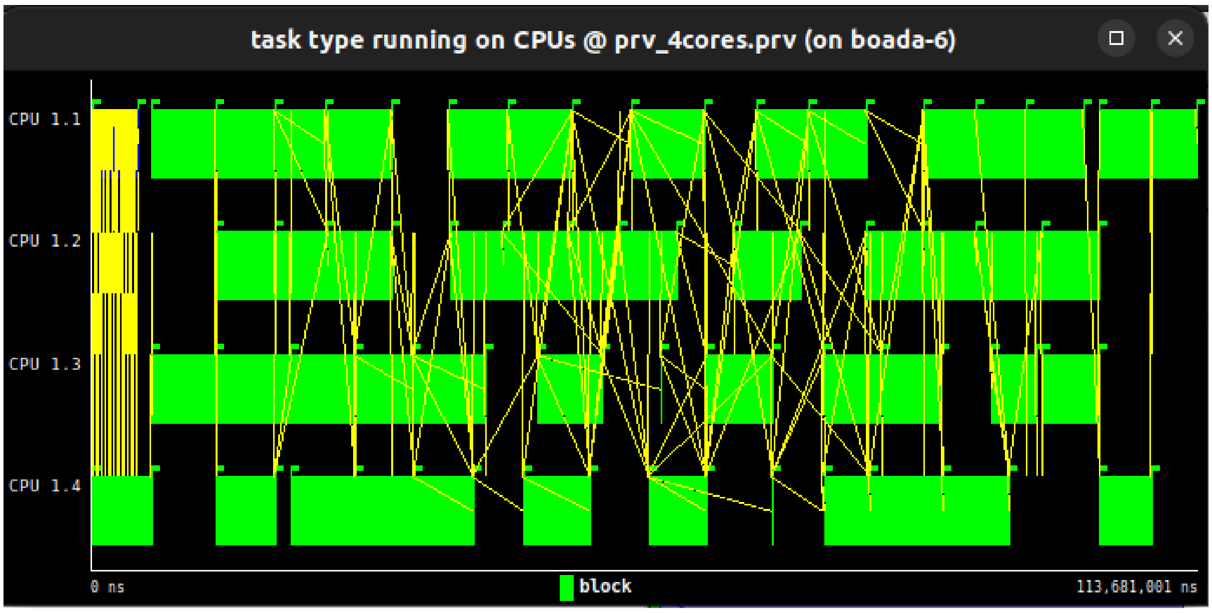


Simulate the execution of both solvers when using 4 threads. Is there any other part of the code that can also be parallelised (take into account this question for the parallel version!)?. If so, modify again the instrumentation to parallelise it.

JACOBI

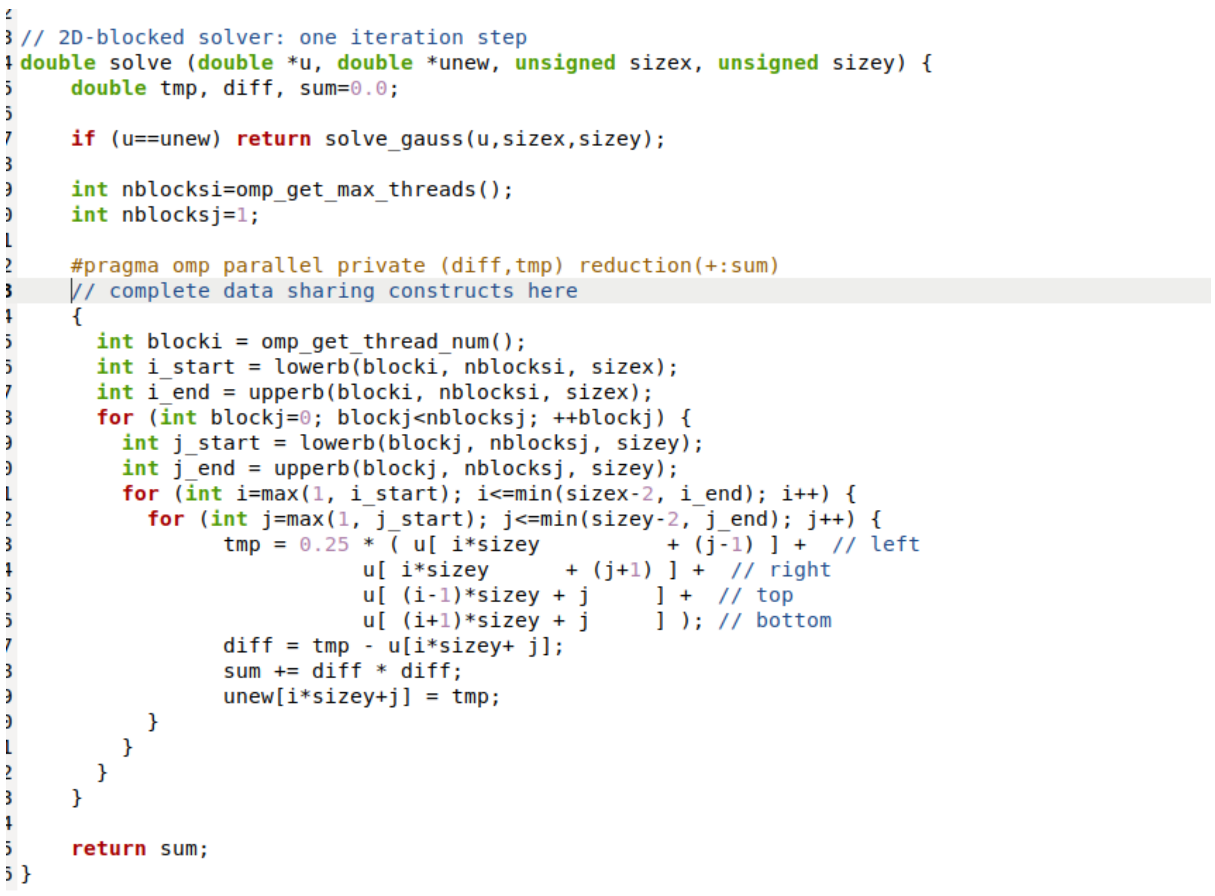


GAUSS

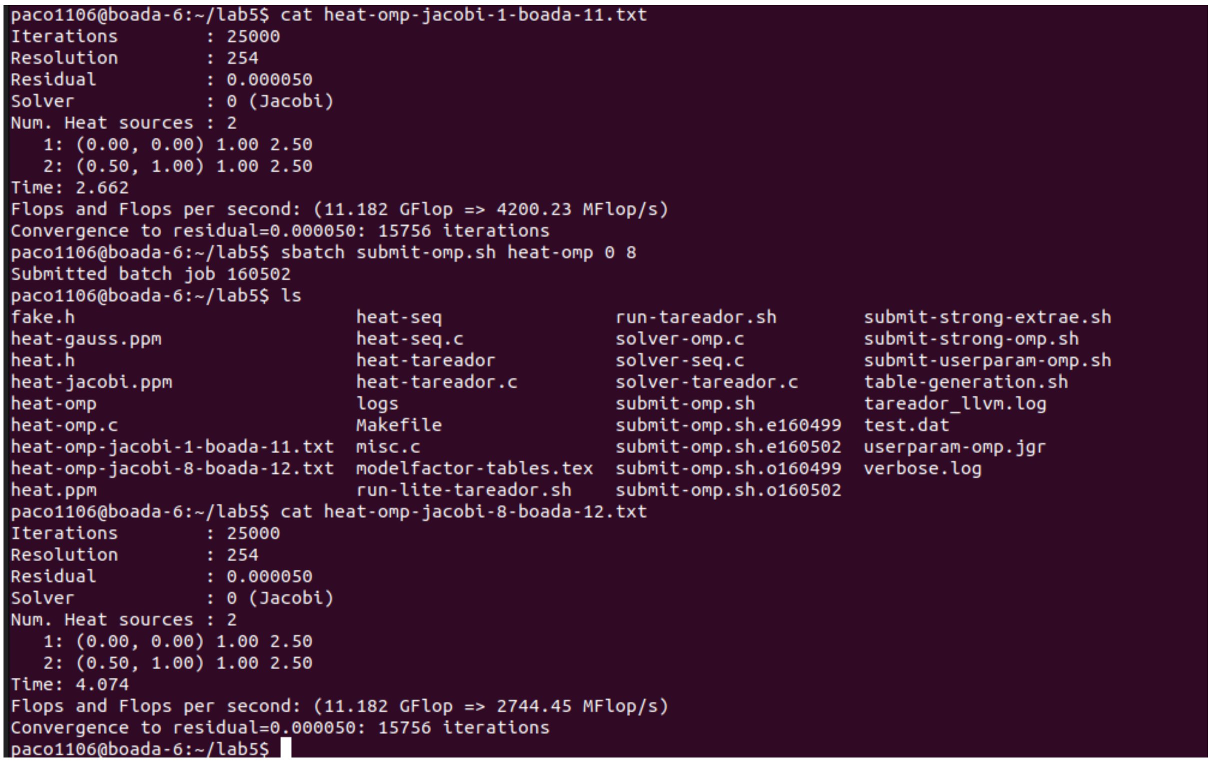


3.Parallelisation of the heat equation solvers

3.1 Jacobi solver



```
// 2D-blocked solver: one iteration step
double solve (double *u, double *unew, unsigned sizex, unsigned sizey) {
    double tmp, diff, sum=0.0;

    if (u==unew) return solve_gauss(u,sizex,sizey);

    int nblocksi=omp_get_max_threads();
    int nblocksj=1;

    #pragma omp parallel private (diff,tmp) reduction(+:sum)
    // complete data sharing constructs here
    {
      int blocki = omp_get_thread_num();
      int i_start = lowerb(blocki, nblocksi, sizex);
      int i_end = upperb(blocki, nblocksi, sizex);
      for (int blockj=0; blockj<nblocksj; ++blockj) {
        int j_start = lowerb(blockj, nblocksj, sizey);
        int j_end = upperb(blockj, nblocksj, sizey);
        for (int i=max(1, i_start); i<=min(sizex-2, i_end); i++) {
          for (int j=max(1, j_start); j<=min(sizey-2, j_end); j++) {
                tmp = 0.25 * ( u[ i*sizey          + (j-1) ] +  // left
                           u[ i*sizey      + (j+1) ] +  // right
                           u[ (i-1)*sizey + j      ] +  // top
                           u[ (i+1)*sizey + j      ] ); // bottom
                diff = tmp - u[i*sizey+ j];
                sum += diff * diff;
                unew[i*sizey+j] = tmp;
          }
        }
      }
    }

    return sum;
}
```

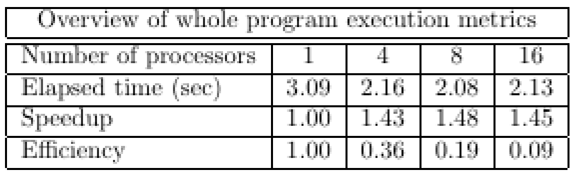


```
paco1106@boada-6:~/lab5$ cat heat-omp-jacobi-1-boada-11.txt
Iterations        : 25000
Resolution        : 254
Residual          : 0.000050
Solver            : 0 (Jacobi)
Num. Heat sources : 2
   1: (0.00, 0.00) 1.00 2.50
   2: (0.50, 1.00) 1.00 2.50
Time: 2.662
Flops and Flops per second: (11.182 GFlop => 4200.23 MFlop/s)
Convergence to residual=0.000050: 15756 iterations
paco1106@boada-6:~/lab5$ sbatch submit-omp.sh heat-omp 0 8
Submitted batch job 160502
paco1106@boada-6:~/lab5$ ls
fake.h                          heat-seq                 run-tareador.sh         submit-strong-extrae.sh
heat-gauss.ppm                  heat-seq.c               solver-omp.c            submit-strong-omp.sh
heat.h                          heat-tareador            solver-seq.c            submit-userparam-omp.sh
heat-jacobi.ppm                 heat-tareador.c          solver-tareador.c       table-generation.sh
heat-omp                        logs                     submit-omp.sh           tareador_llvm.log
heat-omp.c                      Makefile                 submit-omp.sh.e160499   test.dat
heat-omp-jacobi-1-boada-11.txt  misc.c                   submit-omp.sh.e160502   userparam-omp.jgr
heat-omp-jacobi-8-boada-12.txt  modelfactor-tables.tex   submit-omp.sh.o160499   verbose.log
heat.ppm                        run-lite-tareador.sh     submit-omp.sh.o160502
paco1106@boada-6:~/lab5$ cat heat-omp-jacobi-8-boada-12.txt
Iterations        : 25000
Resolution        : 254
Residual          : 0.000050
Solver            : 0 (Jacobi)
Num. Heat sources : 2
   1: (0.00, 0.00) 1.00 2.50
   2: (0.50, 1.00) 1.00 2.50
Time: 4.074
Flops and Flops per second: (11.182 GFlop => 2744.45 MFlop/s)
Convergence to residual=0.000050: 15756 iterations
paco1106@boada-6:~/lab5$ 
```

3.1.2 Overall Analysis

Include the modelfactors tables. Is the scalability that is obtained with this initial parallelisation appropriate? Which is the metric reported by mfLite.py that you should address first?

Veiem que encara es pot paralelitzar si toquem copy_mat



Overview of whole program execution metrics				
Number of processors	1	4	8	16
Elapsed time (sec)	3.09	2.16	2.08	2.13
Speedup	1.00	1.43	1.48	1.45
Efficiency	1.00	0.36	0.19	0.09

Table 1: Analysis done on Sun Dec 24 01:51:07 PM CET 2023, paco1106

Overview of the Efficiency metrics in parallel fraction, ϕ=67.34%				
Number of processors	1	4	8	16
Global efficiency	99.75%	79.15%	67.11%	37.86%
Parallelization strategy efficiency	99.75%	98.99%	98.07%	97.55%
Load balancing	100.00%	99.93%	99.90%	99.89%
In execution efficiency	99.75%	99.06%	98.16%	97.65%
Scalability for computation tasks	100.00%	79.95%	68.43%	38.81%
IPC scalability	100.00%	84.31%	78.10%	51.71%
Instruction scalability	100.00%	96.09%	94.81%	84.30%
Frequency scalability	100.00%	98.69%	92.41%	89.03%

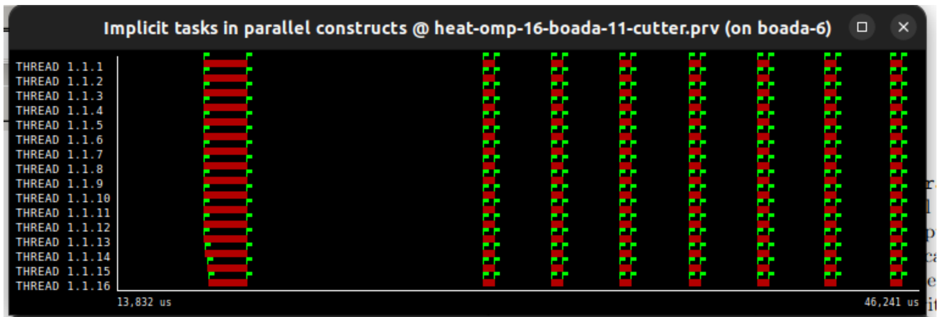
Table 2: Analysis done on Sun Dec 24 01:51:07 PM CET 2023, paco1106

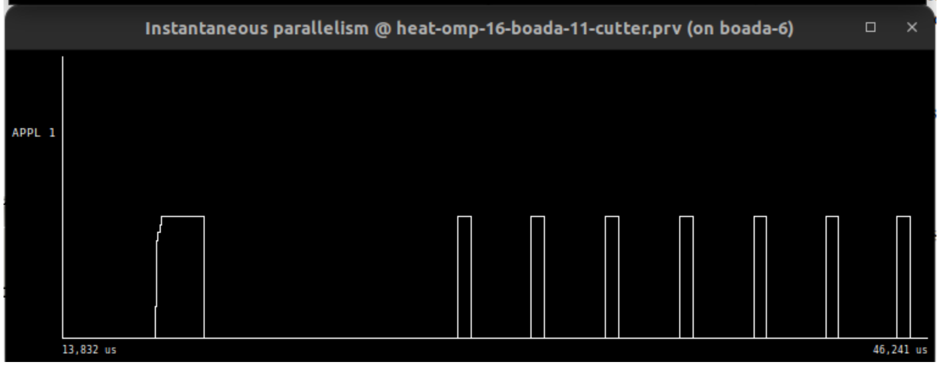
Statistics about explicit tasks in parallel fraction				
Number of processors	1	4	8	16
Number of implicit tasks per thread (average us)	1000.0	1000.0	1000.0	1000.0
Useful duration for implicit tasks (average us)	2076.36	649.23	379.27	334.35
Load balancing for implicit tasks	1.0	1.0	1.0	1.0
Time in synchronization implicit tasks (average us)	0	0	0	0
Time in fork/join implicit tasks (average us)	5.2	0	0	0

Table 3: Analysis done on Sun Dec 24 01:51:07 PM CET 2023, paco1106

3.1.3 Detailed Analysis

Include some captures of the window timelines to show the problem. What is the region of the code that is provoking the low value for the parallel fraction in your parallelisation? Hint: Remember the Tareador analysis you did.



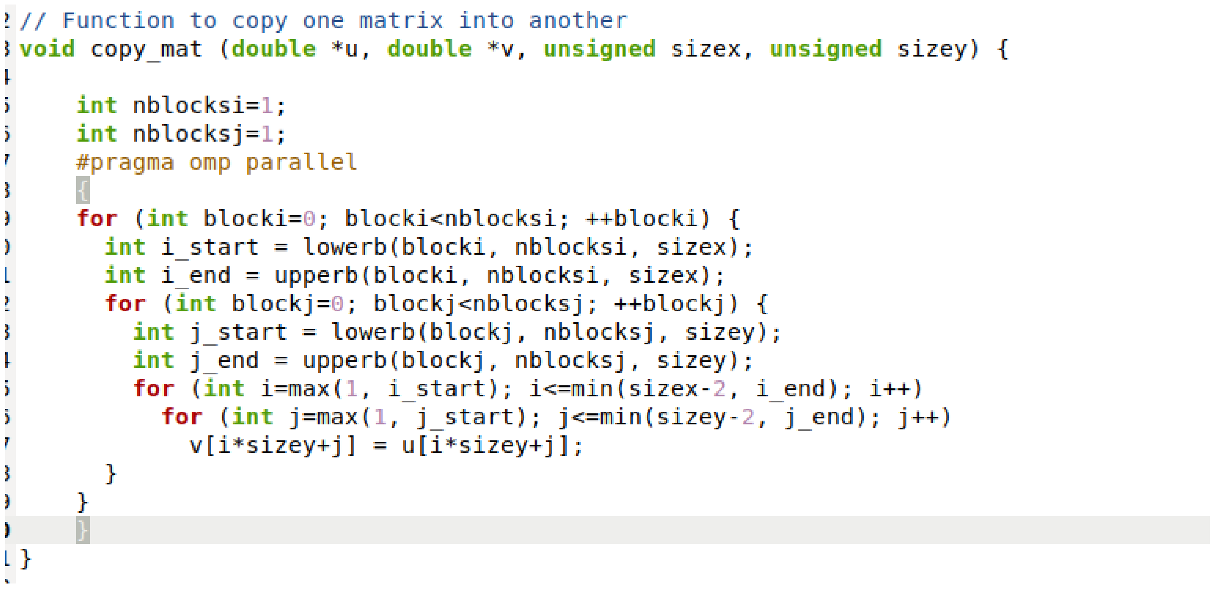

Instantaneous parallelism @ heat-omp-16-boada-11-cutter.prv (on boada-6)
APPL 1
13,832 us
46,241 us

Falta paral·lelitzar copy_mat.

3.1.4 Optimization

Include an excerpt of the code to show the OpenMP annotations you have added to the code.

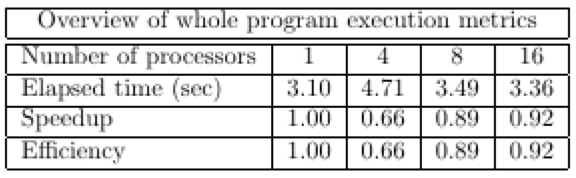
Paral·lelitzem copy_mat



```c
// Function to copy one matrix into another
void copy_mat (double *u, double *v, unsigned sizex, unsigned sizey) {

    int nblocksi=1;
    int nblocksj=1;
    #pragma omp parallel
    {
    for (int blocki=0; blocki<nblocksi; ++blocki) {
      int i_start = lowerb(blocki, nblocksi, sizex);
      int i_end = upperb(blocki, nblocksi, sizex);
      for (int blockj=0; blockj<nblocksj; ++blockj) {
        int j_start = lowerb(blockj, nblocksj, sizey);
        int j_end = upperb(blockj, nblocksj, sizey);
        for (int i=max(1, i_start); i<=min(sizex-2, i_end); i++)
          for (int j=max(1, j_start); j<=min(sizey-2, j_end); j++)
            v[i*sizey+j] = u[i*sizey+j];
      }
    }
    }
}
```

Overall Analysis of the Optimized Code

Include the modelfactors tables and the plot of scalability. Is the execution time reduced?. Have you increased the scalability? Is the parallel fraction larger than before? What is the speedup that you have achieved compared to your first implementation for 16 threads?



Overview of whole program execution metrics				
Number of processors	1	4	8	16
Elapsed time (sec)	3.10	4.71	3.49	3.36
Speedup	1.00	0.66	0.89	0.92
Efficiency	1.00	0.66	0.89	0.92

Table 1: Analysis done on Sun Dec 24 01:41:42 PM CET 2023, paco1106

Overview of the Efficiency metrics in parallel fraction, ϕ=98.89%				
Number of processors	1	4	8	16
Global efficiency	99.72%	65.38%	88.50%	92.02%
Parallelization strategy efficiency	99.72%	76.86%	79.93%	82.78%
Load balancing	100.00%	88.02%	83.19%	85.68%
In execution efficiency	99.72%	87.33%	96.07%	96.62%
Scalability for computation tasks	100.00%	85.06%	110.72%	111.16%
IPC scalability	100.00%	31.56%	31.19%	25.26%
Instruction scalability	100.00%	271.99%	383.32%	486.50%
Frequency scalability	100.00%	99.09%	92.62%	90.44%

Table 2: Analysis done on Sun Dec 24 01:41:42 PM CET 2023, paco1106

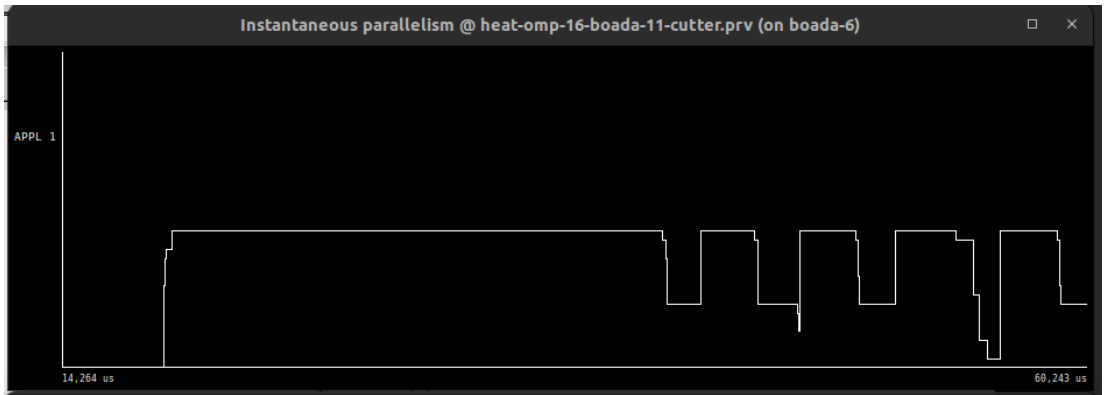
Statistics about explicit tasks in parallel fraction				
Number of processors	1	4	8	16
Number of implicit tasks per thread (average us)	2000.0	2000.0	2000.0	2000.0
Useful duration for implicit tasks (average us)	1529.76	1798.52	1381.61	1376.16
Load balancing for implicit tasks	1.0	0.88	0.83	0.86
Time in synchronization implicit tasks (average us)	0	0	0	0
Time in fork/join implicit tasks (average us)	4.27	0	0	0

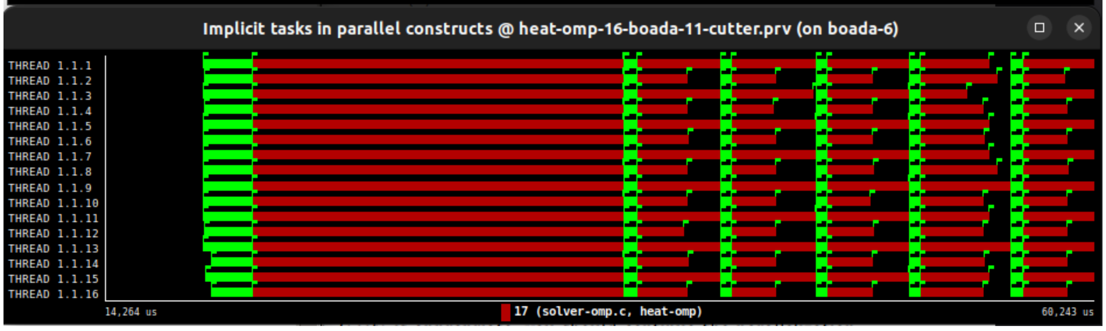
Table 3: Analysis done on Sun Dec 24 01:41:42 PM CET 2023, paco1106

Observer que el temps d'execució és una mica mayor, però millora considerablement l'eficiencia.

Detailed Analysis of the Optimized Code

Include some captures of the window timelines.


Instantaneous parallelism @ heat-omp-16-boada-11-cutter.prv (on boada-6)
APPL 1
14,264 us
60,243 us



3.2 Gauss–Seidel solver

3.2.1 First Implementation

Include an excerpt of the code to show the modifications done.

3.2.2 Overall Analysis

Include the plot obtained with submit-strong-omp.sh. Do you observe a linear speedup? Reason your answer. Note: Remember the analysis done with tareador with a simulation with 4 threads.

3.2.3 Detailed Analysis and Optimization

Include an excerpt of the code to show the modifications done.

3.2.4 Finding the appropriate value for the number of blocks

Include the plots obtained with submit-userparam-omp.sh and submit-strong-omp.sh using the best userparam and 16 threads. Reason why changing the number of blocks in the j dimension changes the ratio between computation and synchronisation. Compare the strong scalability for both cases nblocksj=20 and nblocksj=best userparam*nblocksi (Finding the appropriate value for the number of blocks).