UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona

# PARALLELISM

# DELIVERABLE LAB 5:
# Geometric (data) decomposition using implicit tasks: heat diffusion equation

Alfons Caparrós Bazo **par2305**
Adrià Fuster Martín **par3208**
**Q2 2020-21**

**4 de Juny de 2021**

# TABLA DE CONTENIDOS



no paral·lelitzar bucle de heap
només paral·lelitzar els bucles del solvers
no posarem single ni tasques explícites, tot amb omp parallel .posar privates

amb jacobi no dependencies
amb gauss si

hi var a baix que es pensa que hi han dependencies, (sum, diff) ← tareador_disable_object(punter a la var)
jacobi files senceres

els 2 primers bucles recorren blocs els dos següent l'interior dels blocs
en el jacobi el 2n bucle no ens serveix.

bloc1 hauria d'estar definit per l'id del thread. (per paralelitzar)

# 1. Introduction

In this laboratory assignment we will explore the last decomposition strategy we study in this course: data decomposition making use of implicit tasks. With this strategy the computation that each implicit task has to perform is determined by the data it has to access, either read or write.

This strategy has many benefits in exploiting the location of the data; as each thread will perform the tasks that access the same data. In this lab we will focus our attention on the Geometric ones that are applied to n-dimensional matrice.

The implicit tasks used to express parallelisation strategies based on data decomposition can not synchronise themselves using task dependences. For this reason in this laboratory assignment we will also implement our own synchronisation objects to implement some sort of task ordering constraints.

# 2. Analysis of task granularities and dependences

fer el run.tareador amb el heat 0/1

```
// 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);
    tareador_disable_object(&diff);

    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) {
        tareador_start_task("tmp");
        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;
          }
        }
            tareador_end_task("tmp");

      }
    }
    //tareador_enable_object(&sum);
    //tareador_enable_object(&diff);


    return sum;
}
```

**Image 1.** *solve-tareador* code

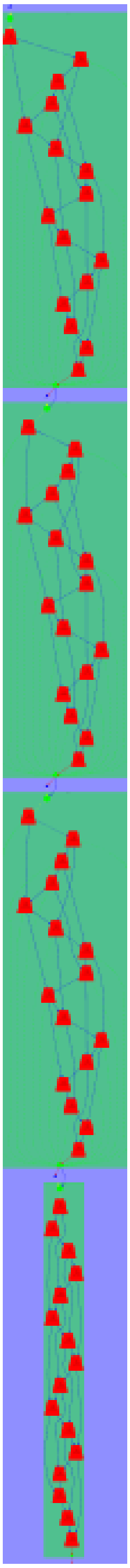


**Image x.** *Gauss* base code

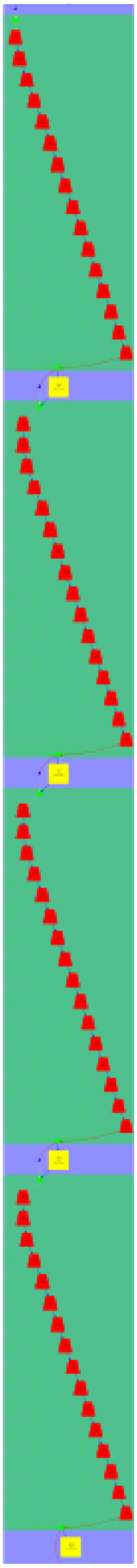
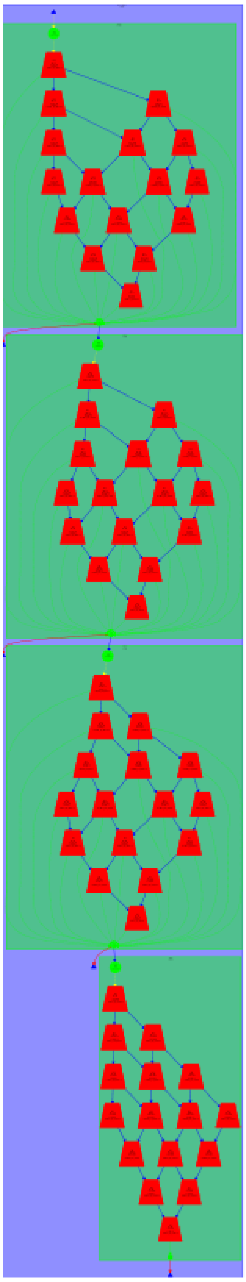


**Image x.** *Jacobi* base code

We can see on images x and x how Gauss TDG has a lot of dependences. Also in Jacobi's case, we checked through Dataview and we realised how the sum variable causes all these dependences. So we disabled this variable and the following TDGs are the result:



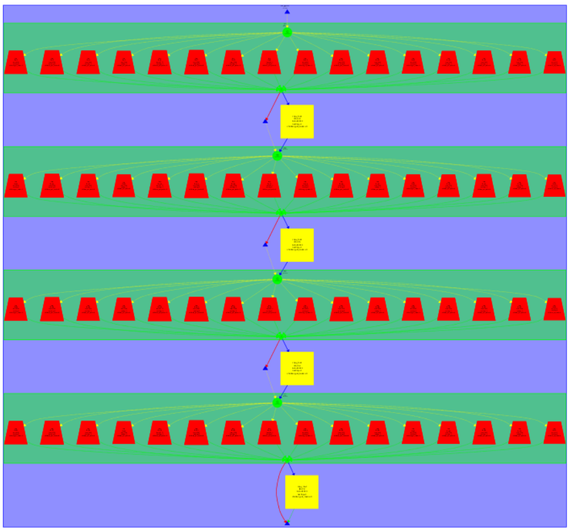


**Image 2.** *Jacobi* with sum disabled

**Image 2.** *Gauss-Seidel* with sum disabled

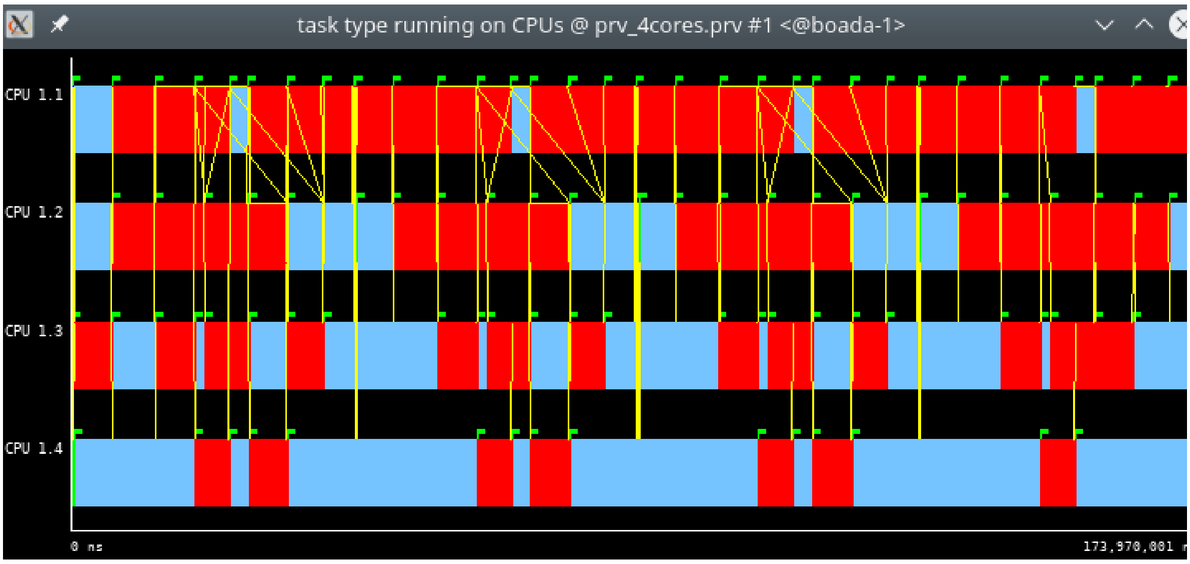



**Image 3.** *Gauss* with sum disabled

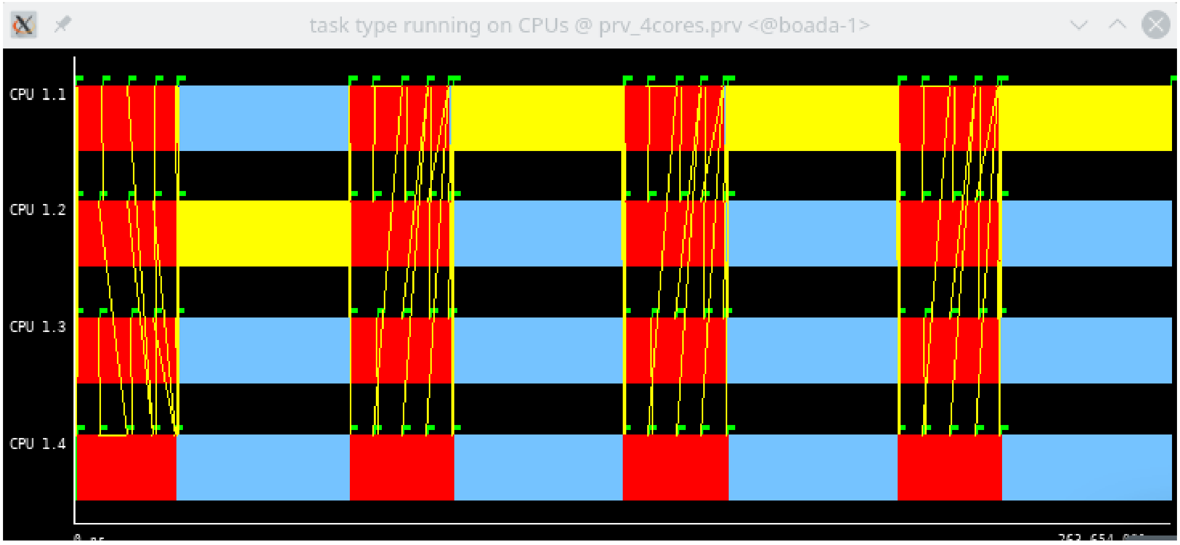



**Image 2.** *Jacobi* with sum disabled

# 3. OpenMP parallelization and execution analysis: *Jacobi*

Editar el solver_omp, i acabar-lo de fer

hem de crar els nostres objectes de sincronitzacio

-vector per mirar quins elem. estan procesats

quan fem unes d'aquestes dues accions fer l'atomic

1.No uc començar a procs un bloc fins que el valor del vector estigui habilitat (barrera) while j < vector[i]

2.Actualitzar el vector per saber quins blocs ja estan fets

PROBLEMA → consistencia de memoria → forçar lectura i flush a mem. amb un atomic update/read

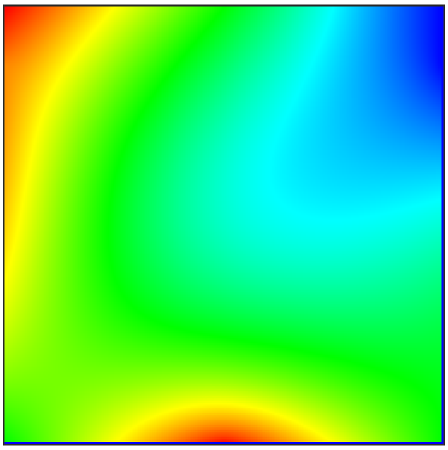
task A escriu tmb next

tasj B llegeix a la var next

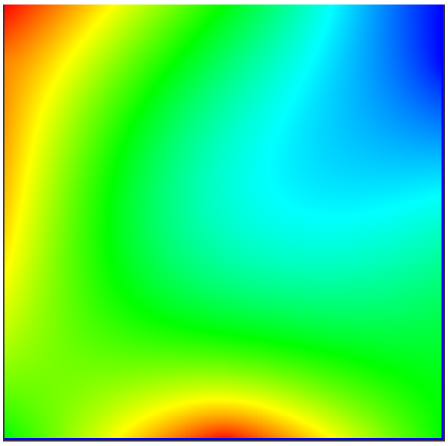
haurem de fer include de les llibreries, ja que farem servir algunes funcions d'elles (OpenMP)

For this part que parallelized the solve function using implicit tasks. As we can see in the following image we declared private the variable and a reduction for the variable `sum`.

```
double solve (double *u, double *unew, unsigned sizex, unsigned sizey) {
    double tmp, diff, sum=0.0;

        #pragma omp parallel private(diff) reduction (+:sum)
      {

            int blocki = omp_get_thread_num();
            int nblocksi = omp_get_num_threads();

            int i_start = lowerb(blocki, nblocksi, sizex);
            int i_end = upperb(blocki, nblocksi, sizex);
            for (int i=max(1, i_start); i<=min(sizex-2, i_end); i++) {
            for (int j=1; j<=sizey-2; 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

                //tareador_disable_object (diff);
                diff = tmp - u[i*sizey+ j];
                //tareador_disable_object (sum);
                sum += diff * diff;
                unew[i*sizey+j] = tmp;
                }
            }
        }

    return sum;
}
```

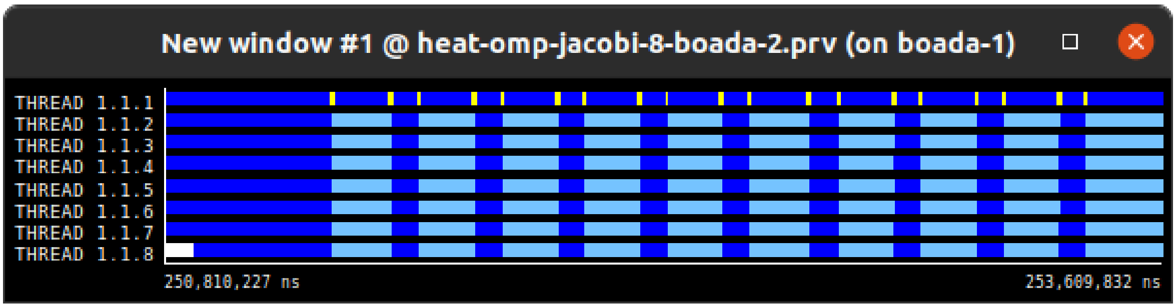


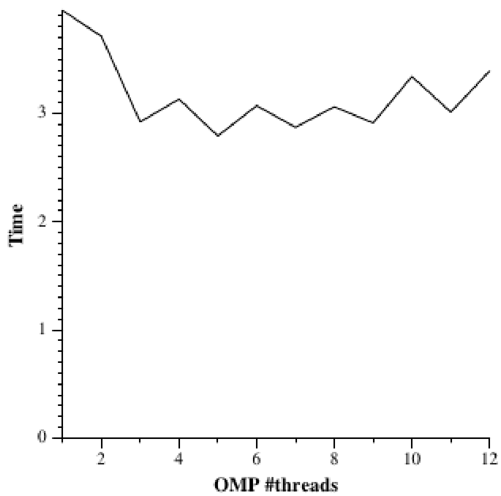
Img x. Jacobi paralelized



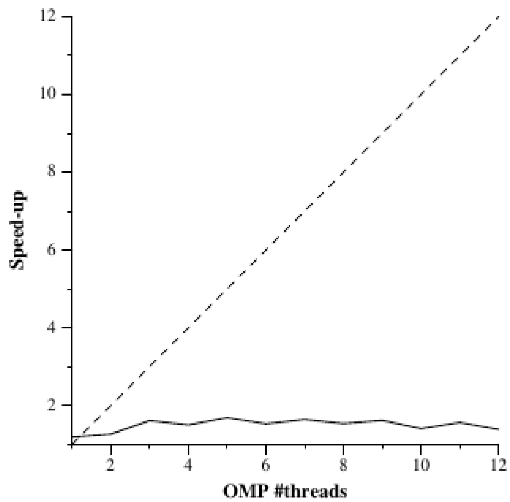
Img x. Jacobi sequential

In order to see the execution of the new parallel code, we displayed the heat image and compared the parallelized with the sequential, and as we can see they look the same.



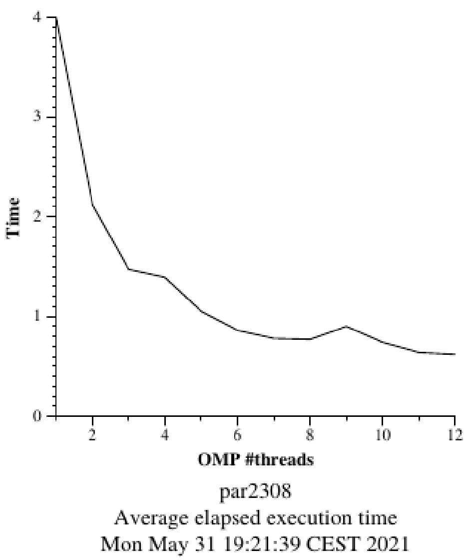

par2308
Average elapsed execution time
Mon May 31 19:11:59 CEST 2021

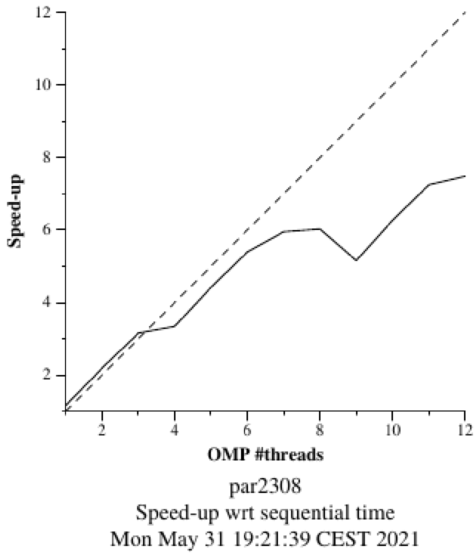



par2308
Speed-up wrt sequential time
Mon May 31 19:11:59 CEST 2021

We can observe that it was not as good as expected. As we increment the number of threads we expect a reduction of the execution time, but it is somehow stable from thread 4 and the speed-up is constant. That is due to the fact that the Jacobi solver uses the `copy_mat` function which was not parallelized.

In order to improve the execution we parallelized the `copy_mat` function too.




Imgx. Strong Scalability plots for Jacobi Solver

As we can see in this speed-up, the performance has increased

# 4. OpenMP parallelization and execution analysis: *Gauss-Seidel*

# 5. Optionals

# 6. Conclusions