David Martos

Mariona Farré

Grupo: paco1208

Curso: 2022-23

PACO:

Laboratorio 5

Index:

[**1. Before starting this laboratory assignment**](#_ascf2opgqezx) **2**

[**2. Sequential heat diffusion program and analysis with Tareador 5**](#_uuj9snud5jz1) **2**

[**3. Parallelisation of the heat equation solvers**](#_g0jl5zoqv2jz) **13**

[3.1. Jacobi Solver](#_at4wh1vo1kl9) 13

[3.1.1. First Implementation](#_fo1i0gcck8ut) 13

[3.1.2. Overall Analysis](#_2tl8tbxz9lm4) 15

[3.1.3. Detailed Analysis](#_e79jfkq5tm1a) 16

[3.2. Gauss-Seidel solver:](#_u9r0aj4k1tys) 19

[3.2.1. First Implementation:](#_dhpvvu8u0a0f) 19

[3.2.2. Overall analysis](#_npj8z0uu5xhh) 20

[3.2.3. Detailed Analysis](#_27ieha7ja5em) 22

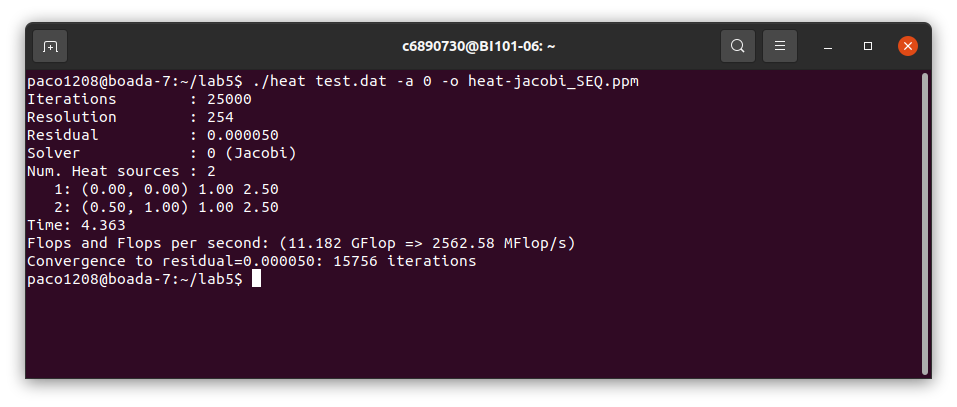
# Before starting this laboratory assignment

# Sequential heat diffusion program and analysis with Tareador 5

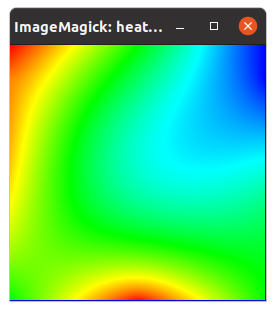
Ejecutar con los solucionadores dados Jacobi y Gauss-Seidel ver su tiempo de ejecución y ver su imagen:

JACOBI:

./heat test.dat -a 0 -o heat-jacobi\_SEQ.ppm

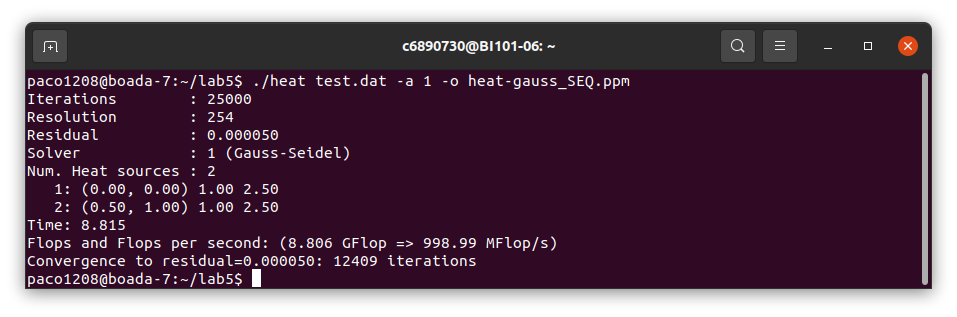


display heat-jacobi\_SEQ.ppm

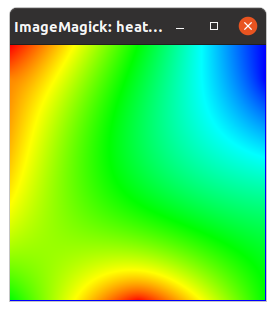


GAUSS:

./heat test.dat -a 1 -o heat-gauss\_SEQ.ppm



display heat-gauss\_SEQ.ppm



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

Para ver si los dos códigos se pueden paralelizar más para obtener una granularidad más fina, debemos ver el gráfico de dependencias original, ver qué tascas tienen dependencias de entrada y salida, y después ver si se puede mejorar creando una nueva tarea dentro del segundo bucle.

Gráficos originales:

| JACOBI: | GAUSS: |
| --- | --- |
|  |  |

**For the deliverable:** Include the excerpt of the code that you have modify in order to specify one task per block.

Poner una tasca por bloque:



Gráficos de tareas con el código inicialmente modificado:

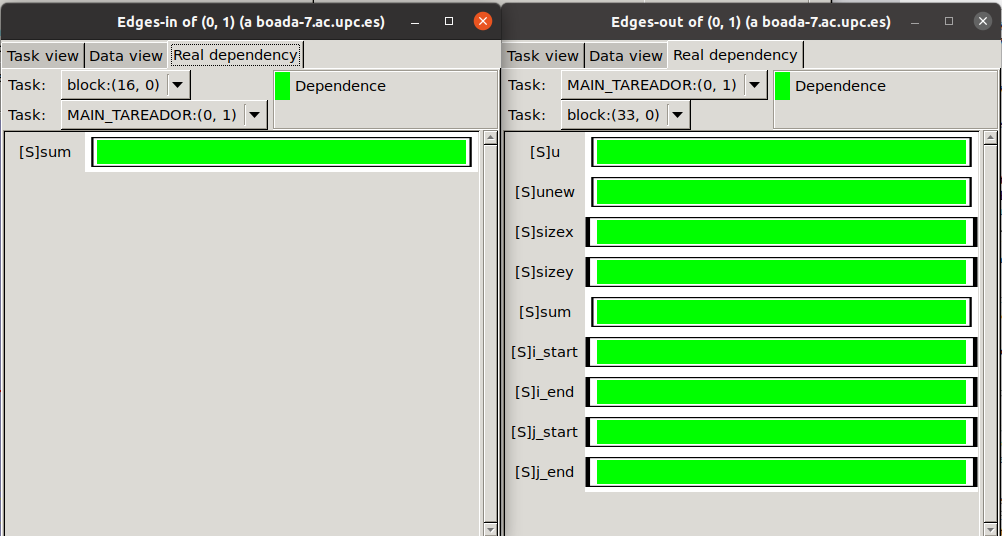
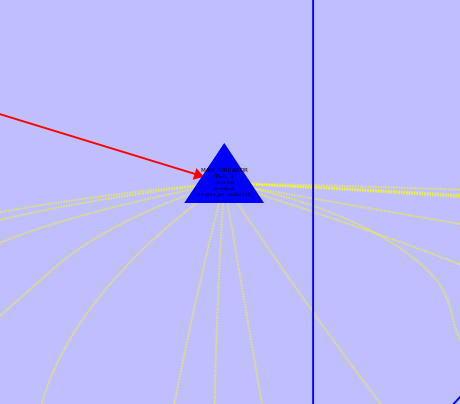
| JACOBI: | GAUSS: |
| --- | --- |
|  |  |

1. Which variable is causing the serialization of all the tasks? Use the **Dataview** option in Tareador to identify it

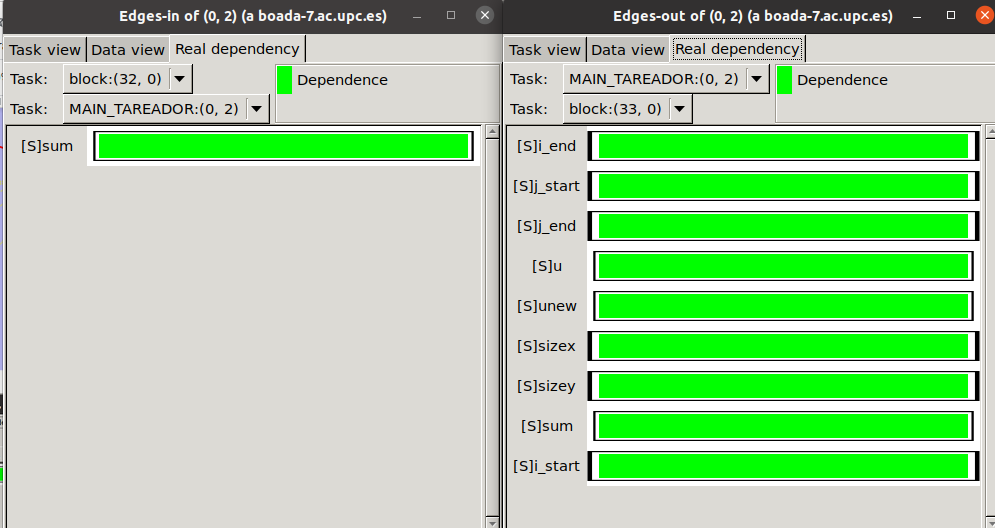
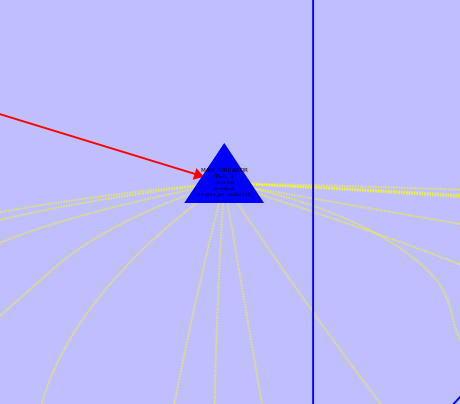
.

Ver que tareas bloquean y crean serialización.

JACOBI:



GAUSS:



1. 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.





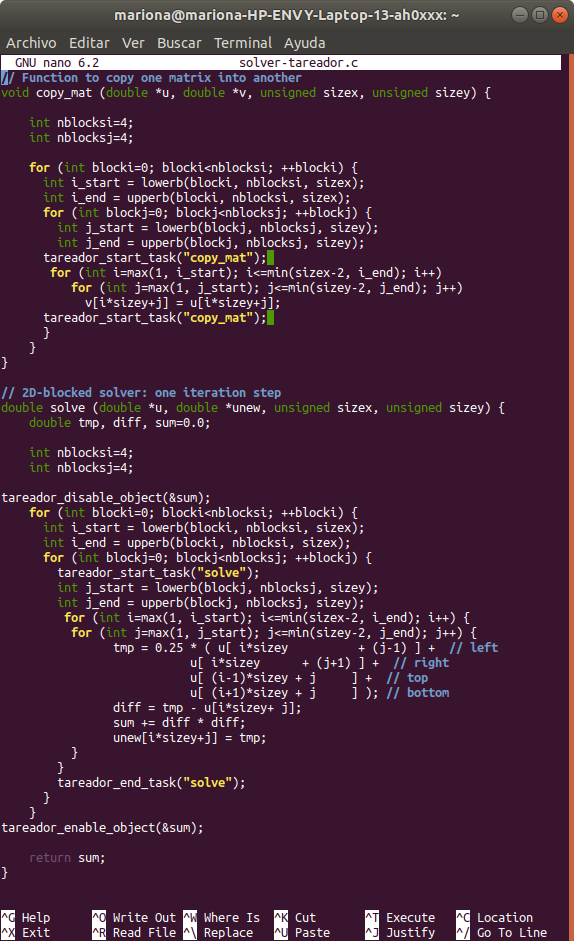
| JACOBI |  |
| --- | --- |
| GAUSS |  |

**For the deliverable:** Which variable was causing the serialisation of all the tasks? Are you obtaining more parallelism? How will you protect the access to this variable in your OpenMP implementation? Simulate the execution when using 4 threads and extract your conclusions. Is there any other part of the code that can also be parallelised?. If so, modify again the instrumentation to parallelise it

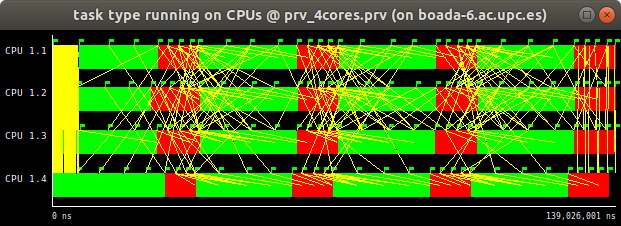
Podemos ver que en los gráficos anteriores, la roja es la tarea que hemos modificado y en verde la tarea del solver ya sea jacobi o gauss que también se modifica, era la tarea en azul la que causaba todos los problemas de serialización.

| JACOBI |  |
| --- | --- |
| GAUSS |  |

Con la ejecución con threads vemos que el verde se paraleliza correctamente, però la funcion de copy-math, que es el hilo amarillo, se pude paraelizar para que sea mas eficiente.



Ejemplo de el heat Jacobi, ha que es el solver que usa el copy mat:



# Parallelisation of the heat equation solvers

## Jacobi Solver

### First Implementation

implicit tasks generated #pragma omp parallel - geometric bloc data decomp by rows

parallelitation function solve

solver-omp.c

complete parallel code jacobi solver

compile: make heat-omp

submit: sbatch submit-omp.sh heat-omp 0 1

diff file generated original seq version

**For the deliverable:** Is the execution time of the OpenMP version of heat using 8 threads reduced

compare to the sequential execution or execution using 1 thread? if not, you should reconsider your

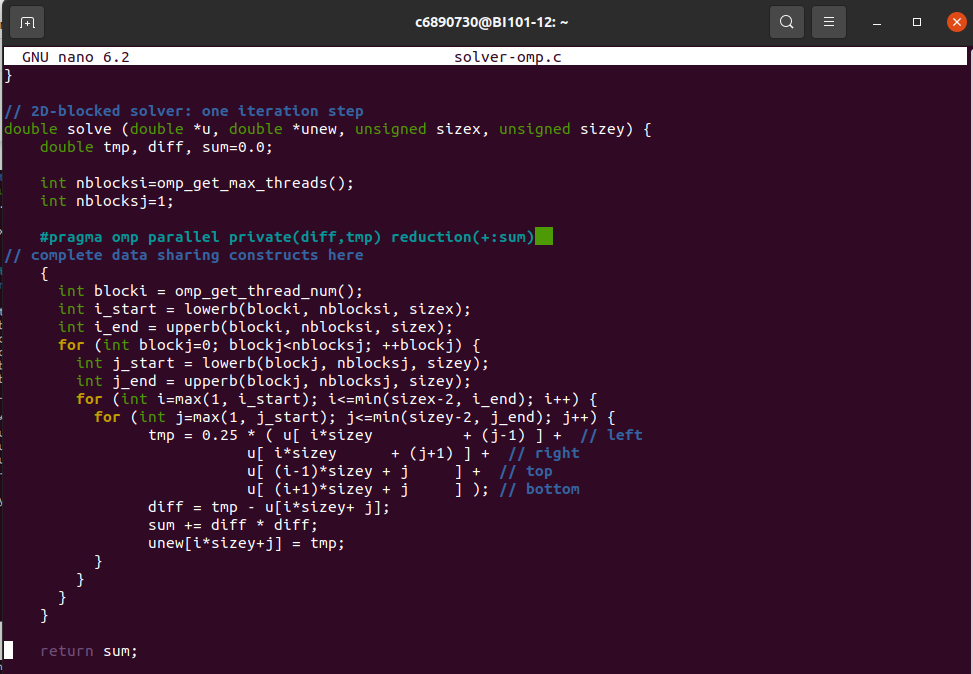
implementaton. For instance, what kind of synchronization are you using? Review different strategies

to avoid and/or reduce the amount of synchronization overheads per interation. Include an excerpt of the code to show the OpenMP annotations you have added to the code.

Se Código original:



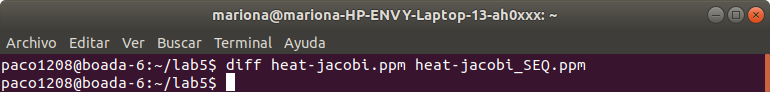
Código con paralelización:



Con 1 thread:



Con 8 threads:

****

Vemos que con 8 threads no da errores comparándolo con el jacobi sequencial diciendo que los dos programas tienen el mismo resultado y esta paralización es correcta.

### Overall Analysis

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

Igualmente que el tiempo mejore, se debe paralizar la función que falta, que sería la función de copy-mat para poder mejorar el código.

| 1 |  |
| --- | --- |
| 4 |  |
| 8 |  |
| 16 |  |

### Detailed Analysis

We recommend you to open with paraver the trace that has been generated for 16 threads and observe what is causing the low value for that metric.

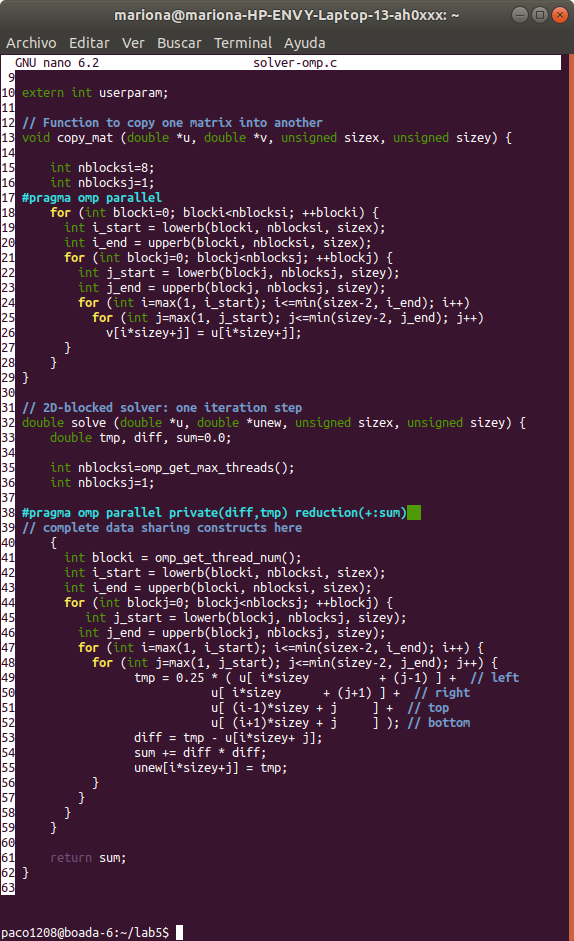
**For the deliverable: In**clude the window timelines or paraver Hints that you consider necessary. What is the region of the code that is provoking the low value for the parallel fraction in your parallelisation?

* + 1. Optimization

Parallelise other parts of the code in order to improve the efficiency of your parallel code. Compile the new version and submit its execution to the queue using the submit-omp.sh script, specifying the binary file, the use of the Jacobi solver and 16 threads. Make sure the new parallel version still generates correct results.

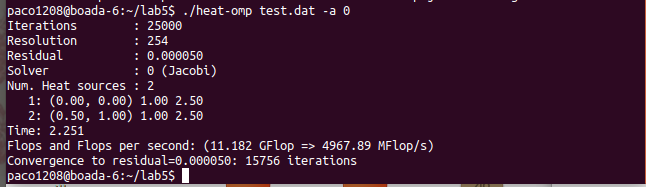
**For the deliverable:** Include an excerpt of the code to show the OpenMP annotations you have added to the code.

Se puede paralelizar la función copy\_mat para que sea mejor su eficiencia, para eso se debe poner un #pragma omp parallel para paralizar la siguiente zona



| 1 |  |
| --- | --- |
| 4 |  |
| 8 |  |
| 16 |  |

JACOBI:



GAUSS:

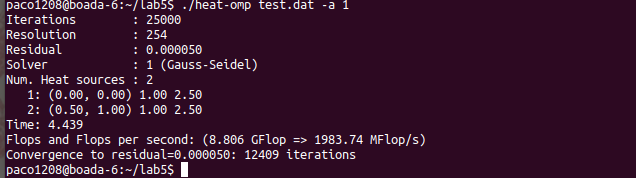
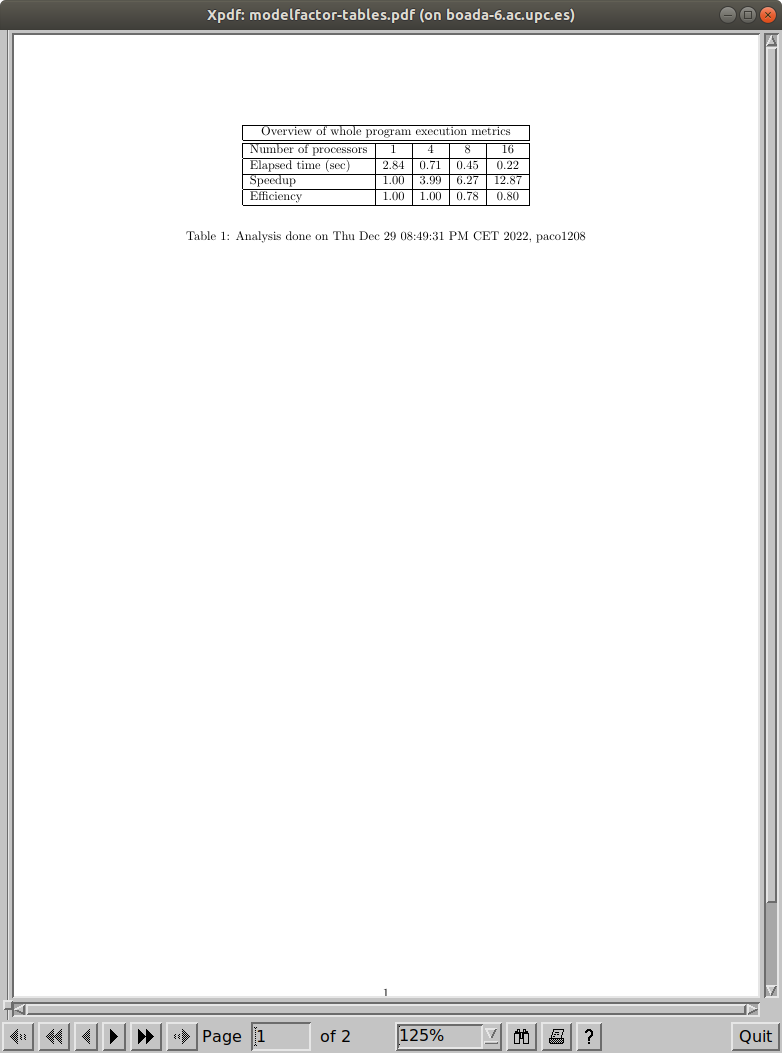


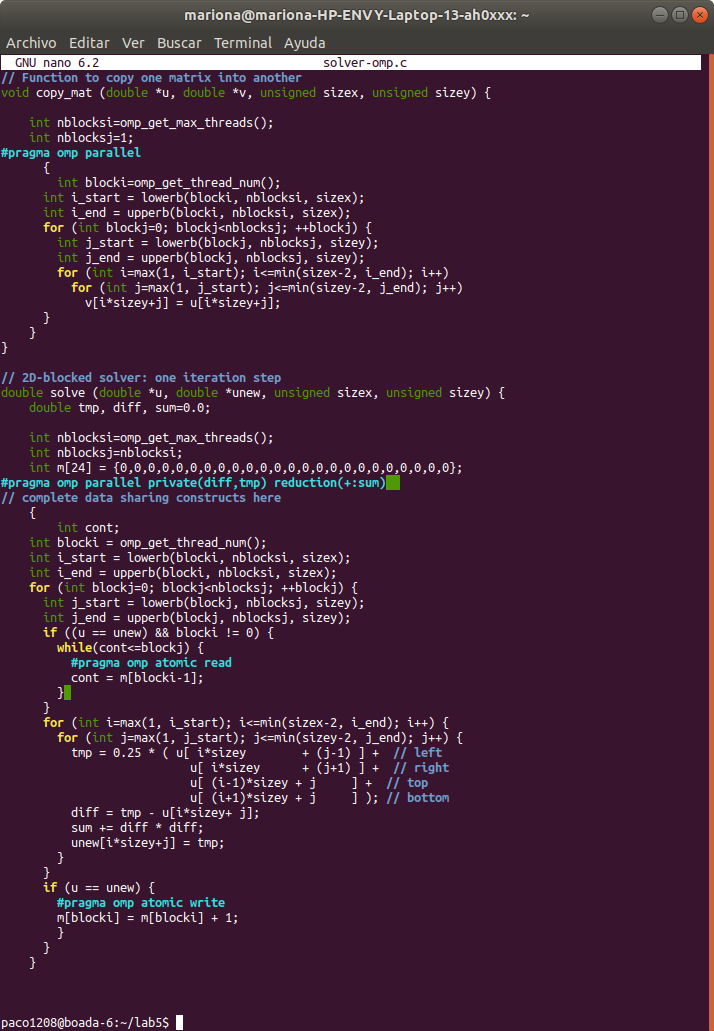
Tabla de modelfactors creada con el código con copy\_mat también paralelizado:

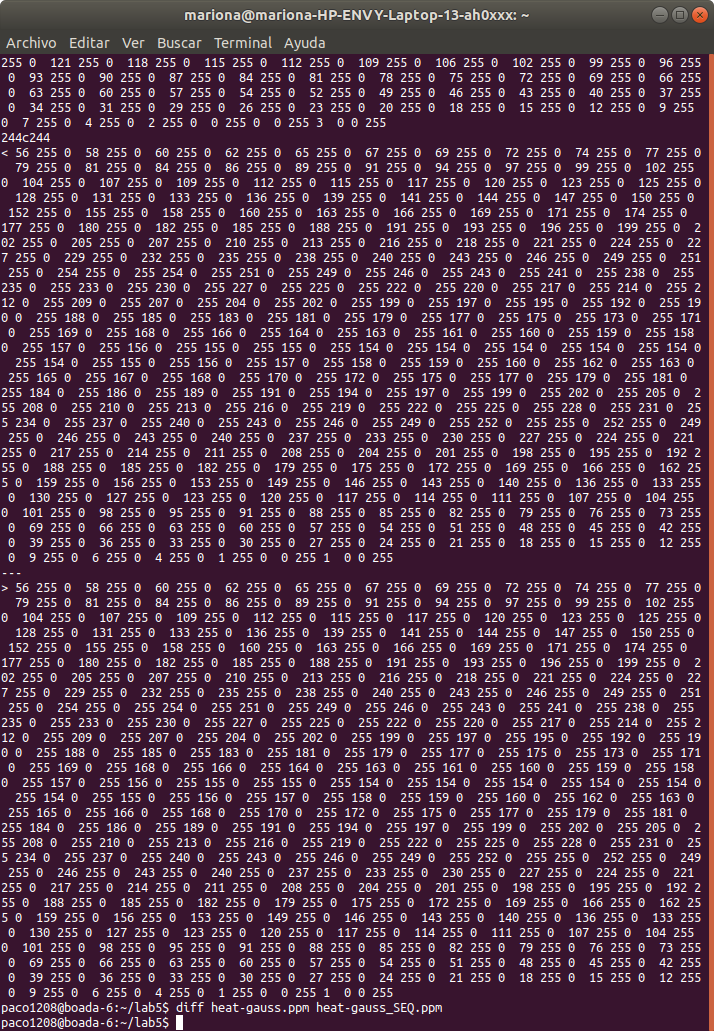
****

## Gauss-Seidel solver:

### First Implementation:

Implementar en la función solve las modificaciones tal y como la documentación nos dice



Al ejecutarse y ser comparado con el resultado sequen cuál de gauss no da ninguna diferencia en la solución, sabiendo que el código modificado continua creando una buena solución:

### Overall analysis

| 1 |  |
| --- | --- |
| 4 |  |
| 8 |  |
| 16 |  |

### Detailed Analysis

**For the deliverable:** Include the Modelfactor tables, the plot of scalability, and the window timelines or paraver Hints that you consider necessary. Is the scalability observed appropriate? Is there any metric reported by modelfactors.py that you should further investigate? Do you think we can increase the parallelism

Podemos observar que hay una menor escalabilidad que en la versión de Jacobi, y comparando los tiempos entre los solvers, la versión de Gauss no le va mejor la paralelización, porque no usa la función copy\_mat como lo hace la versión de Jacobi.

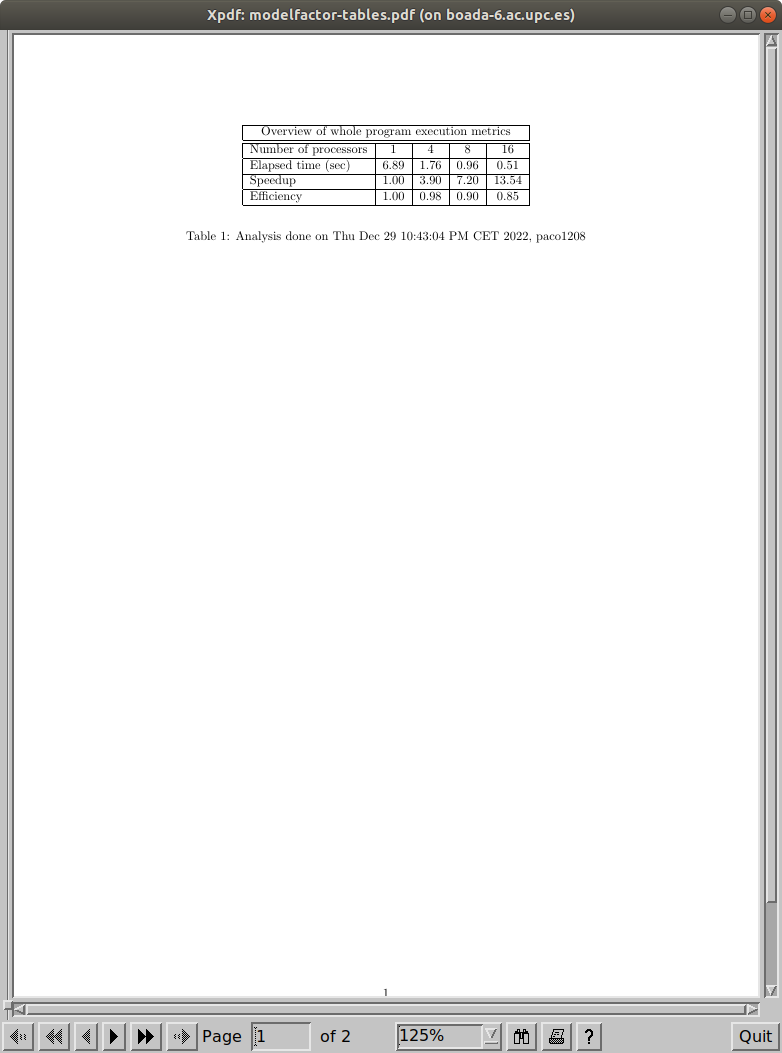
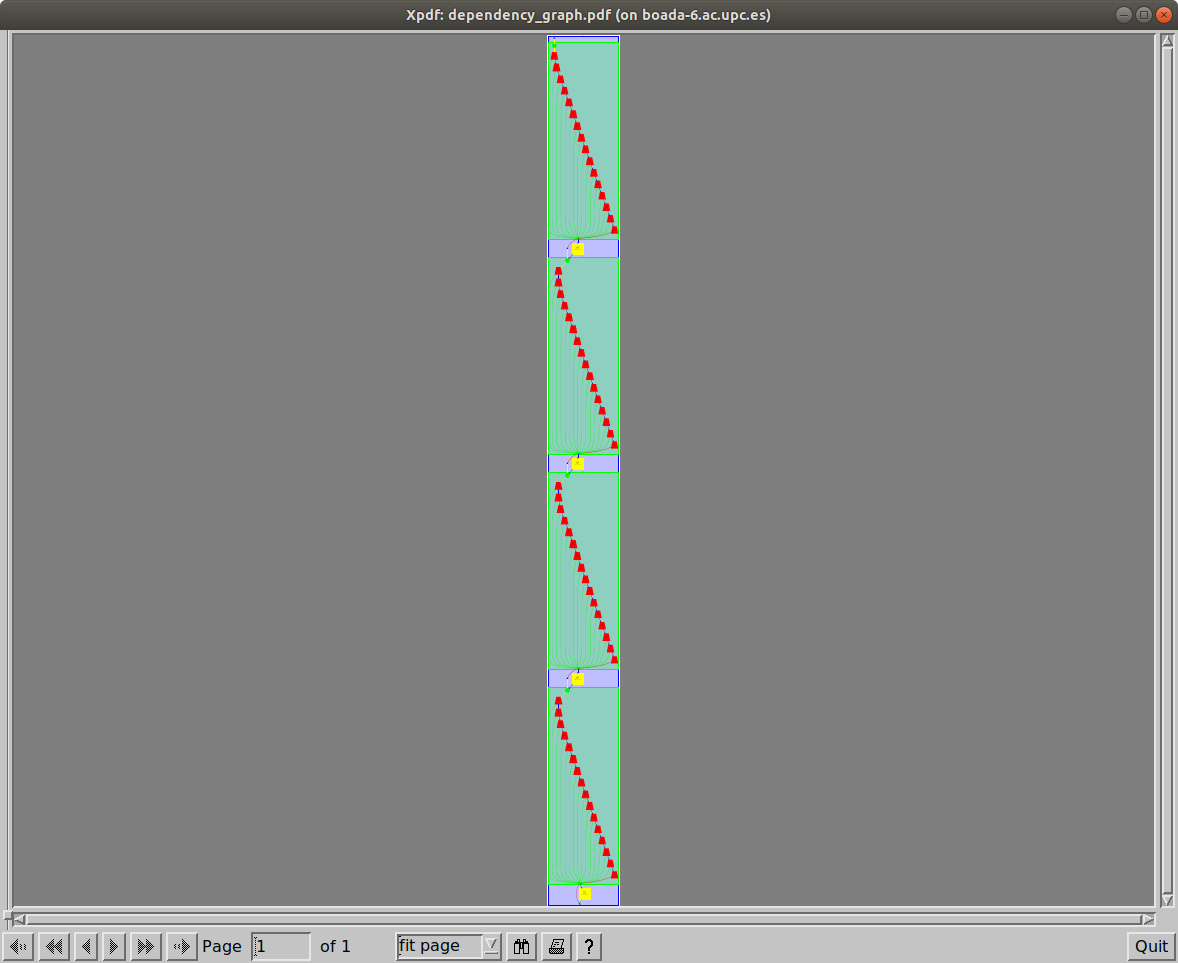


Gráfico de dependencias creado con el código de Gauss modificado:



Mismo gráfico una zona con zoom:

