UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Facultad de Informática de Barcelona
Grado en Ingeniería Informática
Especialidad de Computación

Propagador de ondas acústicas paralelo en 2D/3D basado en el método pseudo-espectral: implementación y análisis de rendimiento

Trabajo de Fin de Grado

Adrián Matas Ruiz

Director: Octavio Castillo Reyes
Tutor GEP: Jorge Enrique Esteban Pérez

18 de junio de 2021

Resumen

El método pseudo-espectral (PSM) es una estrategia numérica utilizada para la solución de ecuaciones de derivadas parciales (PDE) en diversos y variados campos como la dinámica de fluidos, simulación de ondas no lineales y modelización sísmica, entre otros. En el contexto de las aplicaciones geofísicas, los algoritmos basados en el PSM han demostrado mayor precisión y eficiencia en comparación con otros métodos numéricos como el de diferencias finitas (FD). Sin embargo, la implementación eficiente del método PSM requiere de algoritmos óptimos y de estrategias avanzadas de paralelización.

En este proyecto se analiza y mejora el flujo de trabajo de un prototipo PSM para la simulación de ondas acústicas en 2D/3D. En particular, se estudian estrategias computacionales que impacten positivamente en la eficiencia del método numérico y alternativas que mejoren la robustez y flexibilidad del código. El prototipo, escrito en Python, es capaz de generar imágenes geofísicas que representan mapas detallados del interior de la tierra. Estas imágenes y conocimiento tienen aplicación directa en diferentes sectores de los que destacan la monitorización de fuentes de energía geotérmica, la modelización y caracterización de yacimientos de agua, y la propagación de ondas en medios porosos, entre otras.

Resum

El mètode pseudo-espectral (PSM) es una estrategia numérica utilitzada per la resolució d'equacions de derivadas parcials (PDE) en diversos i variats camps com la dinàmica de fluids, simulació d'ones no lineals i modelització sísmica, entre d'altres. En el context de les aplicacións geofísiques, els algoritmes basats en el PSM han mostrat major precisió i eficiencia en comparació amb altres mètodes numérics com el de diferències finites (FD). Però, l'implementació eficient del métodoe PSM requereix d'algoritmes óptims i d'estrategies avançades de paral·lelització.

En aquest projecte s'analitza i es millora el fluxe de treball d'un prototip PSM per la simulació d'ones acústiques en 2D/3D. En particular, s'estudien estrategies computacionals que impacten positivament en l'eficiencia del mètode numéric i alternatives que millorin la robustesa i flexibilitat del codi. El prototip, escrit en Python, es capaç de generar imatges geofísiques que representen mapes detallats de l'interior de la terra. Aquestes imatges i coneixement tenen aplicació directa en diferents sectors dels quals destaquen monitorització de fonts d'energía geotérmica, modelització i caracterització de jaciments d'aigua, i la propagació d'ones en medis porosos, entre d'altres.

Abstract

The pseudo-spectral method (PSM) is a numerical strategy to solve partial differential equations (PDE) applied in diverse and various fields like fluid dynamics, simulation of non-linear waves, and seismic modeling, among others. In the context of geophysical applications, the algorithms based on the PSM have shown greater precision and efficiency compared with other numerical methods such as finite difference (FD). However, an efficient implementation of the PSM method requires optimal algorithms and advanced paralelization strategies.

In this project we analize and improve the main work-flow of a PSM prototype to simulate acoustic waves in 2D/3D. In particular, computational strategies that positively impact efficiecy of the numeric method and alternatives to improve the robustness and flexibility of the code are studied. The prototype, written in Python, is able to generate geophysical images that represent detailed maps of the interior of the earth. These images and knowledge have a direct application in various sectors of which the monitoring of geothermal energy sources, modeling and characterization of water reservoirs, and waves propagation on porus medium, stand out among others.

Índice

Índice de figuras

Índice de cuadros

1. Introducción

1.1. Contextualización

Este Trabajo de Final de Grado (TFG) es de modalidad A, es decir que se ha realizado en la Universidad Politécnica de Catalunya (UPC). Este trabajo está centrado en la especialidad de computación dentro del Grado de Ingeniería Informática de la Facultad de Informática de Barcelona (FIB). Además, el proyecto se ha desarrollado en el *Geosciences Applications Group* del Barcelona Supercomputing Center (BSC)[1]

El proyecto consiste en analizar e implementar mejoras sobre un prototipo PSM de código paralelo escrito en Python. Este código resuelve un sistema de ecuaciones lineales para casos de densidad constante y variable. Este sistema de ecuaciones resulta de la integración en tiempo explicita de la ecuación de la onda acústica en 2D y en 3D. Dependiendo de la integración espacial y temporal se utilizan dos métodos diferentes, el método de Fourier pseudo-espectral y el método de diferencias finitas explicados más adelante. Con todo esto el código es capaz de modelar ondas acústicas en 2D y en 3D. Este código es muy útil para la investigación en el campo de la geofísica.

1.2. Actores implicados

En este trabajo hay diferentes actores implicados:

- **Director del proyecto** El director del proyecto es Octavio Castillo Reyes[2], profesor asociado al departamento de Arquitectura de Computadores de la UPC y a su vez investigador del *Geosciences Applications Group* del BSC[1]. Se encargará de entregar el material necesario para iniciar el proyecto y ayudar al desarrollador en el caso que este la necesite.
- **Desarrollador del proyecto** Es el único desarrollador de este proyecto. Su trabajo consistirá en definir la planificación del proyecto, crear la documentación, analizar y mejorar el código y realizar las pruebas.
- **Usuarios** El resultado del proyecto va dirigido a la comunidad científica interesada en el modelado de ondas acústicas en dos y tres dimensiones (2D/3D).

1.3. Descripción de los problemas

Si bien el método de Fourier PSM es muy eficiente y preciso también tiene sus desventajas. Para poder aplicar este método a la simulación de la propagación de ondas en escenarios realistas y complejos es necesario un gran número de procesadores y una buena optimización de esta parte del algoritmo.

Sin embargo eso no es todo, para poder realizar correctamente los cálculos es necesaria una etapa de interpolación al inicio y al final del proceso. Esta etapa también debe de ser óptima ya que juega un papel esencial en escenarios relativamente grandes.

A pesar de que ya existen soluciones similares para resolver este problema, la mayoría de ellas se ofrecen sin documentación o como cajas negras lo cual restringe no sólo su uso, sino también su verificación y validación así como su adaptación a las necesidades particulares de los usuarios.

1.4. Justificación

Como se ha mencionado anteriormente, existen alternativas para la solución del problema abordado en este TFG. Sin embargo, estas opciones carecen de documentación o de soporte paralelo, lo cual limita su aplicación y uso para la solución de problemas de modelado realistas. Por ello, el desarrollo y mejora del prototipo del modelador inicial queda justificado en el marco de este TFG.

1.5. Objetivo

El objetivo principal del proyecto no solo consiste en analizar un prototipo inicial sino adicionalmente estudiar las diferentes estrategias computacionales que pueden afectar la eficiencia del método y explorar altenativas para mejorar la robutez del código y extenderlo sin sacrificar la flexibilidad. Además el proyecto se desarrollará bajo un enfoque open source con tal de solucionar las restricciones mencionadas en el segundo punto descrito en la sección de problemas.

1.5.1. Subobjetivos

El principal objetivo se divide en los siguentes subobjetivos:

1. Desarrollar prototipo de modelado paralelo con condiciones de frontera
2. Desarrollar un módulo para la entrada y salida de trazas en paralelo
3. Determinar la escalabilidad del código
4. Implementar test unitarios
5. Definir la interfaz del fichero de parámetros y el formato de entrada y salida
6. Crear de un repositorio abierto de fácil descarga y uso

1.6. Alcance

1.6.1. Requerimientos funcionales

Para la solución de problemas de modelado a escala realista, es necesario el uso de tecnologías de computación de alto rendimiento. Por ello, el código a desarrollar en este TFG contará con soporte paralelo y los experimentos se realizarán en el supercomputador MareNostrum4 (MN) del Barcelona Supercomputing Center.

1.6.2. Obstáculos y riesgos

En este proyecto se utilizan librerías externas que permiten añadir nuevas funcionalidades con poco esfuerzo y sin reescribir grandes partes del código. Estas nos pueden resultar de gran ayuda para optimizar el código, pero también podrían suponer una limitación dependiendo del caso a resolver. Por ello se valorará el alcance de cada opción.

1.7. Términos y conceptos

A continuación se describen algunos términos y conceptos relacionados con el proyecto.

1.7.1. Método de fourier pseudo-espectral

Los métodos pseudo-espectrales, o pseudo-spectral methods (PSM) en inglés, son un conjunto de métodos numéricos utilizados en las matemáticas y en las ciencias de la computación para resolver ecuaciones de derivadas parciales (PDE). Estos métodos son aplicados actualmente en dinámica de fluidos, ondas no lineales y modelado sísmico, entre otros.

En el contexto de las aplicaciones geofísicas, los algoritmos basados en estos métodos son priorizados cuando se busca más eficiencia y precisión en comparación con otros métodos como podría ser el método de diferencias finitas (FD). Es un método que tan solo se utiliza en aquellos problemas donde hay una periodicidad natural. En problemas multidimensionales se debe de usar solo en las direcciones donde hay condiciones de frontera periódicas. En nuestro caso se utiliza el método de Fourier PSM para la integración espacial.

1.7.2. Método de diferencias finitas

Las diferencias finitas (FD) es un método de aproximación de derivadas de una función. Juega un rol muy importante para la resolución numérica de ecuaciones diferenciales. Este es el método utilizado para la integración temporal debido a que es una dirección no periódica y no es posible utilizar Fourier PSM

2. Metodología y rigor

En la primera fase del proyecto se realizará una etapa de autoaprendizage y de adaptación al entorno de trabajo. En esta etapa se tendrá que preparar el entorno instalando las dependencias necesarias para poder ejecutar correctamente el prototipo.

A partir de esta etapa se utilizará la metodología ágil Scrum, ya que ésta nos permitirá ser flexibles e ir introduciendo cambios cada cierto tiempo pudiendo así reaccionar a imprevistos. Durante este periodo se definirán ciclos de desarrollo cortos de 2 semanas aproximadamente. Al final de estos ciclos se hará una reunión mediante *Google Meet[3]* donde se pondrá al día de los cambios introducidos en el proyecto al director de éste y se discutirán los siguientes pasos a seguir.

2.1. Herramientas

Una de las herramientas esenciales utilizada en este proyecto es *Git[4]*, un sistema de control de versiones (VCS) que nos permite introducir cambios en el código pero manteniendo las versiones anteriores permitiendo así volver hacia atrás en todo momento. Esta herramienta es muy útil junto con *GitHub[5]*, una página web donde se pueden alojar en repositorios públicos o privados estas versiones y compartirlas con quien se desee. En nuestro caso existe un repositorio privado, con la idea de hacerlo público cuando el proyecto esté finalizado, compartido con el director del proyecto para que así pueda tener un buen seguimiento.

Otra herramienta que se utiliza es *docker[6]* que es un software que automatiza el despliege de aplicaciones en contenedores. Esto permite utilizar la aplicación en un entorno controlado para garantizar al usuario el uso correcto de esta ya que todas las dependencias usarán la versión correcta en este entorno.

Además, se utiliza *Google Meet[3]* para el servicio de videollamadas gratuito. Esta herramienta permite al director crear eventos con fecha y hora para poder encontrarse vía online con el desarrollador y hacer seguimiento. También permite mostrar la pantalla a la vez que se habla lo que resulta muy útil para poder resolver dudas de una forma gráfica

3. Planificación temporal

Con tal de llevar a cabo correctamente el trabajo es necesario hacer una buena planificación temporal. En esta sección se muestra dicha planificación.

Pese a haber matriculado el TFG en febrero, se empezó mucho antes con el trabajo. Concretamente el proyecto comenzó el 13 de Octubre del 2020 y se tiene previsto como fecha límite acabarlo el 28 de Mayo del 2021. En total el trabajo se realiza durante 227 días y se estima que la duración total del trabajo son 540 horas debido a que la carga académica del TFG equivale a 18 créditos (30 horas por crédito).

3.1. Descripción de las tareas

A continuación se ponen en detalle las tareas del proyecto resultantes de los objetivos expuestos anteriormente. Las tareas se distribuyen en diferentes grupos según a la fase del trabajo que correspondan.

3.1.1. Gestión del proyecto (GP)

Este apartado incluye todas las tareas no técnicas relacionadas con la asignatura de Gestión de Proyectos (GEP).

Alcance y contexto (GP1)
Esta fase pretende revisar el estado del arte y poner el proyecto en contexto de la FIB y definir su alcance especificando los objetivos, subobjetivos y requisitos del trabajo. Su duración se estima que sean 25 horas.

Planificación temporal (GP2)
En esta tarea se propone una planificación temporal inicial donde se definen las tareas a realizar durante el TFG y se distribuyen en el tiempo con tal de contar con un plan desde el principio. Se estima que esta tarea dure 15 horas.

Presupuesto y sostenibilidad (GP3)
La tarea pretende definir los costes del proyecto con un presupuesto así como al escrito de un informe de sostenibilidad económica, ambiental y social. Su duración se estima que sean 15 horas.

Memoria (GP4)
Como su nombre indica, consiste en escribir la memoria final del proyecto donde consta la documentación de todas las fases del proyecto así como la conclusión final. Se prevee que tenga una duración de 80 horas.

Presentación (GP5)
Una vez finalizada la escritura de la memoria, ésta se presentará ante el tribunal del TFG. En esta fase se creará el material de la presentación como su guión y sus ensayos. La duración de la tarea es de 20 horas.

Reuniones (GP6)
Durante todo el proyecto se cuenta con reuniones con el director para ponerse al día y discutir cambios o mejoras. Se prevee que estas reuniones se hagan periódicamente cada dos semanas aproximadamente y que la duración de cada una sea de 1 hora. En total contamos con 16 horas.

3.1.2. Trabajo previo (TP)

Preparación del entorno de trabajo (TP1)
Para poder ejecutar el código correctamente es necesario contar con entorno en el equipo. Concretamente se necesita una distribución de Linux que en nuestro caso es *Ubuntu18.04 LTD [7]* y además es necesario instalar las dependencias de la aplicación. Esta tarea se estima que sea de 25 horas.

3.1.3. Desarrollo e implementación de mejoras (DM)

Estudio de sistema de control de versiones (DM1)
En esta tarea se realizará un estudio del funcionamiento de *Git[4]* y *GitHub[5]* con el objetivo de crear correctamente un repositorio para compartir el código con el director del proyecto y usarlo adecuadamente durante toda la trayectoria del trabajo. La duración de la tarea es de 10 horas.

Estudio e implementación de docker (DM2)
En esta tarea se realizará un estudio del funcionamiento de *docker[6]* y la implementación de una imagen que contenga todas las dependencias con sus correctas versiones y el código de la aplicación. Esto nos permite garantizar al usuario que podrá utilizar correctamente la aplicación. La duración de la tarea es de 20 horas.

Revisión, análisis y ejecuciones del código en paralelo (DM3)
Esta tarea pretende estudiar el "workflow", la entrada y salida de datos y las estructuras de datos utilizadas en el código. También se observan los tiempos de ejecución y el consumo de memoria del mismo para la simulación en casos relevantes. Tanto los datos como el estudio previo son necesarios recopilarlos en un documento escrito. La duración de esta tarea es de 40 horas.

Refactorización del código (DM4)
Una vez estudiado el código esta tarea consiste en el estudiar e introducir varias mejoras en el "workflow"principal de la aplicación. Esta tarea será realizada en varios sprints de la metodología ágil de Scrum. La duración de esta tarea se estima que sea de 135 horas.

Estudio y implementación de las optimizaciones (DM5)
Una vez finalizado el desarrollo e implementación de las funcionalidades básicas del código, se iniciará la fase de optimización. En esta tarea se estudiará las diferentes herramientas para conseguir un código limpio y óptimo (Numba,Cython, etc). La duración de esta tarea es de 80 horas.

Ejecuciones en paralelo y análisis de rendimiento (DM6)
Con una versión final del código se realizará un análisis del rendimiento con varias ejecuciones en paralelo comparando esta versión con la versión del código no optimizado. Su duración es de 35 horas.

Empaque (DM7)
Finalmente, el código se subirá al repositorio del proyecto. Éste incluirá la versión final del código, su documentación y algunos casos de uso. La duración de la tarea se estima que sea de 20 horas.

3.2. Recursos

A continuación se muestran todos recursos necesarios para el proyecto.

- **Humanos (R1)**: El director del proyecto (R1d) y programador (R1p).
- **Materiales (R2)**: El ordenador personal del programador ya que toda tarea ya sea de gestión o del desarrollo se realiza desde casa.

- **Software (R3)**: Todo el software utilizado es gratuito, entre ellos Git/GitHub, Thonny para la edición de código, docker para la gestión de dependencias y LaTeX para la composición de documentos de texto.

Id	Tarea	Tiempo	Dependencias	Recursos
GP	**Gestión del proyecto**	**175h**	**-**	**-**
GP1	Alcance y contexto	25h	-	R1,R2
GP2	Planificación temporal	15h	GP1	R1,R2
GP3	Presupuesto y sostenibilidad	15h	GP1	R1,R2
GP4	Memoria	80h	-	R1p,R2
GP5	Presentación	20h	GP4	R1p,R2
GP6	Reuniones	20h	-	R1,R2
TP	**Trabajo Previo**	**25h**	**-**	**-**
TP1	Preparación del entorno de trabajo	25h	-	R1p,R2
DM	**Desarrollo e implementación de mejoras**	**340h**	**-**	**-**
DM1	Estudio de sistema de control de versiones	10h	TP1	R1p,R2,R3
DM2	Estudio e implementación de docker	20h	TP1	R1p,R2,R3
DM3	Revisión, análisis y ejecuciones del código en paralelo	40h	TP1,DM1	R1p,R2,R3
DM4	Refactorización del código	135h	DM3	R1p,R2,R3
DM5	Estudio e implementación de las optimizaciones	80h	DM4	R1p,R2,R3
DM6	Ejecuciones en paralelo y análisis de rendimiento	35h	DM5	R1p,R2,R3
DM7	Empaque	20h	DM6	R1p,R2,R3
-	**Total**	**540h**	**-**	**-**

Cuadro 1: *Tabla de tareas de la planificación inicial con su duración, dependencias y recursos. Elaboración propia.*

3.3. Representación gráfica de la planificación

Nombre	Duración	Inicio	Fin
Preparación del entorno de trabajo	9días?	10/13/2020	10/23/2020
Alcance y contexto	6días?	10/13/2020	10/20/2020
Planificación Temporal	6días?	10/21/2020	10/28/2020
Presupuesto y sostenibilidad	6días?	03/08/2021	03/15/2021
Memoria	142.5días?	10/14/2020	04/30/2021
Presentación	21días?	05/03/2021	05/31/2021
Desarrollo e implementación de mejoras	134días?	**11/02/2020**	**05/06/2021**
Estudio de sistema de control de versiones	11días?	11/02/2020	11/16/2020
Estudio e implementación de docker	11días?	11/16/2020	11/30/2020
Revisión, analisis y ejecuciones del código en paralelo	10días?	12/01/2020	12/14/2020
Refactorización del código	35días?	12/15/2020	02/01/2021
Estudio y implementación de las optimizaciones	23días?	02/02/2021	03/04/2021
Ejecuciones en paralelo y analisis de rendimiento	23días?	03/05/2021	04/06/2021
Empaque	22días?	04/07/2021	05/06/2021

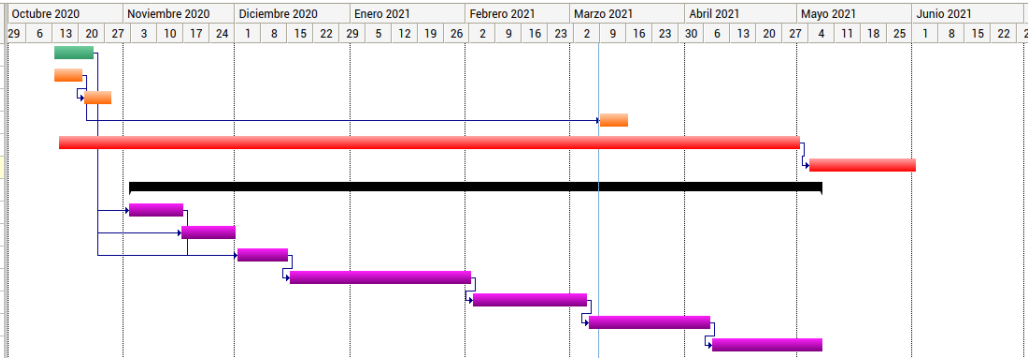


Figura 1: *Diagrama de Gantt creado con Gantter. Elaboración propia.*

3.4. Gestión del riesgo

Es muy importante preveer los riesgos que se pueden dar durante el proyecto así como contar con alternativas para garantizar que el trabajo finaliza dentro del plazo. Por esa razón a continuación se nombran posibles riesgos con sus alternativas.

1. Pequeños errores - Estos se tienen en cuenta durante el desarrollo gracias a la metodología Scrum ya que nos permitirá rectificarlos inmediatamente después de detectarlos. Se estima que estos errores consuman 10 horas de trabajo adicionales.
2. Recursos limitados - Como ya se comentó anteriormente, es posible que las librerías utilizadas por la aplicación nos limiten. Para ello se necesitará buscar otras librerías alternativas o incluso desarrollar la funcionalidad necesaria de propia mano. Por ello se cuenta con 1 més extra de margen antes de la entrega para poder rectificar pero se estima que esta nueva tarea dure entre 15 y 20 horas.
3. Averías de recursos materiales - Debido a que el ordenador personal del programador tiene 6 años y su vida útil ronda los 8 años es bastante probable de que sufra una avería. Si se da el caso, el programador cuenta con un ordenador portátil que puede sustituir al ordenador principal. Debido a que el ordenador portátil es mucho menos potente que el ordenador personal puede relentizar el trabajo. Por este motivo contaremos con 15 horas más.

3.5. Cambios respecto a la planificación inicial

En el momento en que esto es escrito, el proyecto se encuentra en la tarea *DM6 - Ejecuciones en paralelo y análisis de rendimiento*. Por motivos de falta de tiempo y de dificultad a la hora de implementar ciertas funcionabilidades han habido los siguientes cambios respecto a la planificación inicial.

3.5.1. Tareas

La decisión de volver el repositorio público de la tarea *DM7 - Empaque* pasa a estar fuera del trabajo y será decisión del *Geosciences Applications Group*

del BSC. Por ello el subobjetivo *6.Crear de un repositorio abierto de fácil descarga y uso* pasaría a ser *6.Crear de un repositorio de fácil descarga y uso con posiblidad de convertirse en abierto.*

Id	Tarea	Tiempo	Dependencias	Recursos
GP	**Gestión del proyecto**	**175h**	**-**	**-**
GP1	Alcance y contexto	25h	-	R1,R2
GP2	Planificación temporal	15h	GP1	R1,R2
GP3	Presupuesto y sostenibilidad	15h	GP1	R1,R2
GP4	Memoria	80h	-	R1p,R2
GP5	Presentación	20h	GP4	R1p,R2
GP6	Reuniones	20h	-	R1,R2
TP	**Trabajo Previo**	**17h**	**-**	**-**
TP1	Preparación del entorno de trabajo	17h	-	R1p,R2
DM	**Desarrollo e implementación de mejoras**	**378h**	**-**	**-**
DM1	Estudio de sistema de control de versiones	10h	TP1	R1p,R2,R3
DM2	Estudio e implementación de docker	23h	TP1	R1p,R2,R3
DM3	Revisión, análisis y ejecuciones del código en paralelo	50h	TP1,DM1	R1p,R2,R3
DM4	Refactorización del código	155h	DM3	R1p,R2,R3
DM5	Estudio e implementación de las optimizaciones	85h	DM4	R1p,R2,R3
DM6	Ejecuciones en paralelo y análisis de rendimiento	40h	DM5	R1p,R2,R3
DM7	Empaque	15h	DM1	R1p,R2,R3
-	**Total**	**570h**	**-**	**-**

Cuadro 2: *Tabla de tareas de la planificación final con su duración, dependencias y recursos. Elaboración propia.*

Por lo que hace al resto de tareas se mantienen pero han sufrido cambios en la duración de algunas y en los plazos de tiempo mostrados en el Cuadro 2.

Finalmente la duración del trabajo es por lo tanto de 570h. Esto afectará en los costes del proyecto explicados más detalladamente en la sección *6.3.Cambios respecto al presupuesto inicial*. En la siguiente sección se muestra el diagrama de Gantt de la planificación actual.

3.5.2. Representación gráfica de la planificación actual

Nombre	Duración	Inicio	Fin
Preparación del entorno de trabajo	5días?	10/19/2020	10/23/2020
Alcance y contexto	6días?	10/13/2020	10/20/2020
Planificación Temporal	6días?	10/21/2020	10/28/2020
Presupuesto y sostenibilidad	6días?	03/08/2021	03/15/2021
Memoria	163días?	10/14/2020	05/28/2021
Presentación	20días?	05/03/2021	05/28/2021
Desarrollo e implementación de mejora	154días?	**10/27/2020**	**05/28/2021**
Estudio de sistema de control de veriones	8días?	10/27/2020	11/05/2020
Estudio e implementación de docker	13días?	10/30/2020	11/17/2020
Revisión, analisis y ejecuciones del código en paralelo	23días?	11/17/2020	12/17/2020
Refactorización del código	65días?	12/18/2020	03/18/2021
Estudio e implementación de las optimizaciones	35días?	02/19/2021	04/08/2021
Ejecuciones en paralelo y analisis del rendimiento	30días?	04/19/2021	05/28/2021
Empaque	145días?	11/09/2020	05/28/2021

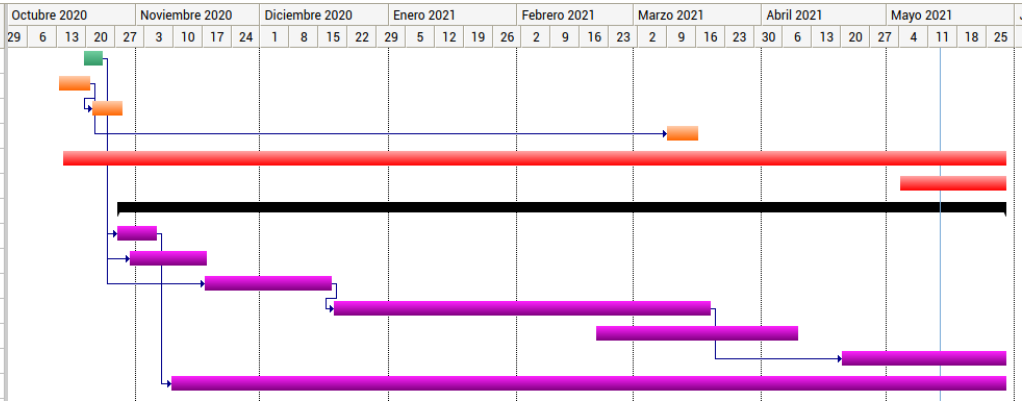


Figura 2: *Diagrama de Gantt creado con Gantter. Elaboración propia.*

4. Presupuesto

4.1. Identificación de los costes

Los costes del proyecto se pueden clasificar en los siguientes tipos:

- **Personales**: Director y programador del proyecto.
- **Directos**: Ordenador personal y entorno de los trabajadores.
- **Indirectos**: Consumo eléctrico del personal y coste del acceso a internet.

4.1.1. Personales

Para calcular los costes de cada persona implicada en el proyecto se cuenta con *"Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública"[8].* Teniendo en cuenta que la jornada máxima laboral es de 1800 horas anuales podemos realizar los cálculos representados en la siguente tabla.

Perfil	**Grupo**	**Nivel**	**Salario bruto(€)**	**SS(€)**	**Coste/año**	**Coste/h**
Director	A	-	26790,31	8037,09	34827,40	19,35
Programador	C	1	24640,37	7392,11	32032,48	17,80

Cuadro 3: *Tabla de los cálculos de los costes personales. Elaboración propia.*

Una vez realizados estos cálculos es posible utilizarlos para calcular los Costes Por Actividad (CPA).

Id	Tarea	Director	Programador	Coste(€)
GP	**Gestión del proyecto**	**25h**	**150h**	**3153,75**
GP1	Alcance y contexto	5h	20h	452,75
GP2	Planificación temporal	5h	10h	274,75
GP3	Presupuesto y sostenibilidad	5h	10h	274,75
GP4	Memoria	-	80h	1424
GP5	Presentación	-	20h	356
GP6	Reuniones	10h	10h	371,50
TP	**Trabajo Previo**	**-**	**25h**	**445**
TP1	Preparación del entorno de trabajo	-	25h	445
DM	**Desarrollo e implementación de mejoras**	**-**	**340h**	**6052**
DM1	Estudio de sistema de control de versiones	-	10h	178
DM2	Estudio e implementación de docker	-	20h	356
DM3	Revisión, análisis y ejecuciones del código en paralelo	-	40h	712
DM4	Refactorización del código	-	135h	2403
DM5	Estudio y implementación de las optimizaciones	-	80h	1424
DM6	Ejecuciones en paralelo y análisis de rendimiento	-	35h	623
DM7	Empaque	-	20h	356
-	**Total (CPA)**	**25h**	**515h**	**9650,75**

Cuadro 4: *Tabla de los cálculos de los costes personales por actividad. Elaboración propia.*

4.1.2. Directos

Son una parte de los costes materiales que no van relacionados con ninguna tarea en concreto. Se pueden diferenciar en dos clases diferentes:

- **Hardware**: Contamos con el ordenador personal (PC) del programador
- **Software**: Se contará con software como LaTeX, Git, docker y Python. Todas estas herramientas son software libre y de código abierto.

Elemento	Coste(€)	Vida Útil(meses)	Coste Amortizado(€)
PC	2128,22	96	98,39
LaTeX	0	0	0
Git	0	0	0
docker	0	0	0
Python	0	0	0
Total	**2128,22**	**-**	**98,39**

Cuadro 5: *Tabla de los costes directos. Elaboración propia.*

La amortización del PC se calcula con un coeficiente de amortización del 25 % mostrado en la *tabla de coeficientes de amortización lineal de la agencia tributaria[9].* Teniendo en cuenta el coeficiente, la vida útil del PC que coincide con el periodo máximo de amortización de 8 años y que la duración del proyecto se estima en 540 horas obtendriamos $(2128,22{*}0{,}25{*}540)/(8{*}365) = 98{,}39$

4.1.3. Indirectos

También son parte de los costes materiales. Se dividen en dos específicos que son:

- **Consumo eléctrico**: Teniendo en cuenta que el consumo máximo del ordenador personal es de 750W pero contando que la mayoría del tiempo tiene una carga del 40 %, contamos que su consumo medio es de 300W. Esto que equivaldría a 118,50€ anuales. Por lo tanto, el coste de éste durante el proyecto sería $(118,50 * 540)/(8 * 365) = 21,91$€
- **Acceso a internet**: La factura mensual del acceso a internet es de 48,76€/mes. El coste de acceso a internet durante todo el proyecto sería de $(48,76 * 12 * 540)/(8 * 365) = 108,21$€

4.1.4. Contingencia

En todo proyecto es importante contar con una contingencia para tener en cuenta los contratiempos y complicaciones que se pueden dar durante este. Los valores en proyectos de software rondan entre un 10 % y un 20 %, en nuestro caso escogeremos un valor medio de 15 %.

Tipo	Coste(€)	Contingencia(€)
Personales	9650,75	1447,61
Directos	98,39	14,76
Indirectos	130,12	19,52
Total	**9879,26**	**1481,89**

Cuadro 6: *Tabla de las contingencias. Elaboración propia.*

4.1.5. Imprevistos

Debido a los posibles riesgos que puede tener el proyecto se cuenta con el plan de un mes extra de trabajo. Lo que sería igual a extender el proyecto 50 horas. También contamos con una probabilidad de un 20 % de que esto ocurra y por tanto el coste de los imprevistos sería:

$$50 * 17,80 * 0,2 = 178€$$

4.1.6. Total

Para acabar, en esta sección se recogen todos los costes del proyecto en la siguente tabla.

Tipo	Coste(€)
Personales	9650,75
Directos	98,39
Indirectos	130,12
Contingencia	1481,89
Imprevistos	178
Total	**11539,15**

Cuadro 7: *Tabla de los costes totales. Elaboración propia.*

4.2. Control de gestión

Una vez definido el presupuesto inicial se definen métodos para establecer un control de posibles desviaciones de los costes dadas durante el proyecto y que nos serán útiles para poder rectificarlas.

En el caso de los costes personales por tarea pueden variar debido al incumplimiento del plazo de la tarea o incluso al precio de la mano de obra. Es por eso que se contará con la siguiente fórmula para determinar su desvío:

$$d_p = (coste\_estimado - coste\_real) * horas\_reales$$

Respecto a los costes directos, si hubiera un fallo en el hardware sería posible con un ordenador portátil de posesión del programador. Por lo que al software se refiere al ser todo de código abierto y libre no habría ninguna desviación.

En cambio para los costes indirectos al variar con las horas se utilizará otro método para calcular su desviación. Éste es el siguente:

$$d_i = (horas\_estimadas - horas\_reales) * coste\_estimado$$

4.3. Cambios respecto al presupuesto inicial

El hecho de haberse visto modificada la duración de las tareas en la planificación final influye directamente en el presupuesto. El aumento de 30h del trabajo del Programador se ve reflejado en:

- **costes personales** que pasarían de **9650,75€** a **10184,75€**
- **costes indirectos** que pasarían de **130,12€** a **137,35€**

Una diferencia en total de **541,23€** que quedaría más que cubierta con la contingencia establecida de **1481,89€**

5. Informe de sostenibilidad

Después de haber contestado la encuesta del proyecto EDINSOST me he dado cuenta de que, a pesar de personalmente querer construir un mundo mejor e incluso haber formado parte de proyectos con ese objetivo, respecto a proyectos TIC tengo un nivel más bajo de lo que me gustaría.

En estos proyectos resulta mucho más intuitivo tener en cuenta la dimensión económica durante el desarrollo de éste, en cambio la ambiental y la social ni siquiera se plantean desde un inicio. Considero la sostenibilidad de un proyecto muy importante para tener en mente el impacto que tendrá tanto el desarrollo del producto como el producto final en las diferentes dimensiones.

Por último, creo que todos los trabajos deberían valorar la sostenibilidad de sus proyectos y me resulta muy importante la realización de un informe como éste para así reflexionar sobre el impacto que ambiental, económico y social y aprender de ello.

5.1. Dimensión ambiental

En ningún momento se ha estimado el impacto ambiental durante el desarrollo del proyecto, sin embargo este proyecto no supone una gran amenaza al medio ambiente ya que durante todo el desarrollo se ha utilizado un único ordenador y también los recursos utilizados en el repositorio de github que son compartidos entre todos los usuarios. Por lo que el desarrollo del código tiene un impacto medio ambiental muy reducido.

Por lo que hace a las soluciones ya existentes pueden que utilizen servidores exclusivamente para su solución o incluso la fabricación del material para distribuir el software que serán residuos en un futuro.

5.2. Dimensión económica

En cuanto al presupuesto de los costes económicos del proyecto es cierto que ha habido una tendencia a sobreestimar los costes ya que había que contar con riesgos que es posible que no se den. Sin embargo, se tenían en cuenta estas probabilidades para ajustarse a una situación real.

Teniendo en cuenta de que el producto final será un software libre, lo que lo diferencia de otras soluciones, por ello desde un punto de vista de usuario económicamente no puede ser mejor ya que es gratuito. Sin embargo, a nivel de empresa como ya se ha visto anteriormente el proyecto tiene un coste económico y al ser software libre estos costes no se recuperarán nunca.

5.3. Dimensión social

El proyecto me aportará conocimientos técnicos en el ambito de paralelismo y análisis y optimización de algoritmos. Conocimientos sobre lenguajes de programación y High Performance Computing. Por otro lado también me aportará conocimientos sobre organización, gestión del tiempo o comunicación con el director del proyecto.

Por lo que hace a las ventajas de este proyecto en el campo social podemos comentar lo que supondrá para los investigadores en el campo de la geofísica. Podrán realizar experimentos en sus investigaciones de forma eficiente. A su vez, estos experimentos serán de utilidad en el mundo.

5.4. Matriz de sostenibilidad

Se analizará la sostenibilidad del proyecto con la matriz de sostenibilidad. Cuenta con las tres dimensiones mencionadas anteriormente y las etapas de Proyecto Puesto en Producción (PPP), su vida útil y sus riesgos. Lo evaluaremos con un número del 0 al 10, de menos a más sostenible.

	PPP	Vida útil	Riesgos
Ambiental	consumo del dieño: 9	huella ecologica: 7	ambientales: 9
Económico	factura: 6	plan de viabialidad: 7	económicos: 9
Social	impacto personal: 9	impacto social: 9	sociales: 8

Cuadro 8: *Matriz de sostenibilidad del proyecto. Elaboración propia.*

6. Aspectos legales

Este trabajo únicamente está regulado por una ley llamada propiedad intelectual.

6.1. Propiedad intelectual

Tal como indica la *Normativa del TFG[10]*:

"La propiedad industrial e intelectual de los TFG de modalidad A está regulada por la normativa aprovada por el Consejo de Gobierno(10/10/2008) por la qual se aprueba la confidencialidad, responsabilidad patrimonial y propiedad indistrial e intelectual a la UPC.

- Corresponderá a la UPC la titularidad sobre las invenciones desarrolladas exclusivamente por los estudiantes si se ha desarrollado en el marco de una actividad académica que ha estado dirigida y/o coordinada por el profesorado de la UPC.
- En el caso que el desarrollo de la obra intelectual haya estado dirigida y/o coordinada por el profesorado de la UPC, corresponderá a la UPC la titularidad de los derechos de explotación sobre esta obra y el estudiante y el profesor serán considerados coautores de la misma.
- En el caso de explotación de la obra por parte de la UPC que le suponga un beneficio económico, el autor o conjunto de autores tendrán derecho a una participación del 50 % de los beneficios netos obtenidos."

6.2. Licencias

Todas las licencias del software utilizado para el desarrollo de este proyecto son de código abierto y libre.

7. Propagador de ondas acústicas 2D/3D (PSM)

La precisión y eficiencia de las técnicas de imágenes geofísicas como inversión de forma de onda completa (FWI) o migración de tiempo inverso (RTM) dependen en gran medida de las técnicas numéricas utilizadas para modelar la propagación de ondas sísmicas. En la aproximación acústica, las fases se pueden reproducir correctamente mediante la ecuación de onda escalar. Existen muchos métodos para modelar la propagación de ondas en medios acústicos. Los métodos más comunes son el explícito en el tiempo y el orden superior en diferencias finitas espaciales (FD). A pesar del amplio éxito de este método, la aproximación precisa y eficiente de las derivadas de tiempo y espacio continua siendo un tema abierto al estudio. Por un lado, tradicionalmente la integración de tiempo se obtiene utilizando esquemas de FD de bajo orden. Por otro lado, la precisión espacial podría mejorarse mediante el uso de diferenciación espectral. El método de dominio de tiempo pseudoespectral de Fourier (PSTD) se basa en dicha diferenciación espectral por medio de la transformada de Fourier (TF). Dadas sus ventajas en precisión numérica, el código utilizado y mejorado en este TFG se basa en el método PSTD.

El problema físico considerado en este trabajo está governado por la ecuación acústica

$$\frac{\partial^2 u(\mathbf{x},t)}{\partial t^2} + \mathcal{L}^2 u(\mathbf{x},t) = 0 \tag{1}$$

donde $u(\mathbf{x},t)$ es la presión acústica en el tiempo t. El operador espacial se define de acuerdo a $\mathcal{L} = lc\nabla$ donde $l = \sqrt{-1}$.

Se considera un dominio computacional $\mathbf{x} \in \Omega$ en 2D y 3D a través del cual la velocidad del sonido $c(\mathbf{x})$ podría variar.

Para el término espacial continuo de la ecuación 1, el término $\mathcal{L}^2$ puede expandirse como

$$\mathcal{L}^2 u(\mathbf{x},t) = -c^2(\mathbf{x})\{\mathcal{F}_x^{-1}[k_x^2 \mathcal{F}_x[u(\mathbf{x},t)]] + \mathcal{F}_y^{-1}[k_y^2 \mathcal{F}_y[u(\mathbf{x},t)]] + \mathcal{F}_z^{-1}[k_z^2 \mathcal{F}_z[u(\mathbf{x},t)]]\} \tag{2}$$

donde k_x, k_y, y k_z son el número de onda en cada dimensión cartesiana; $\mathcal{F}$ y $\mathcal{F}^{-1}$ denotan la TF continua en 1D y su inversa, respectivamente. La formulación de los métodos de Fourier PS se basa en un análogo discreto de la ecuación 2, en el que las derivadas 1D se calculan en el dominio del número de onda mediante TF discreto (DFT) y se devuelven al dominio espacial mediante una TF inverso. Por ello, se ha considerado un medio rectangular Ω discretizado por una cuadrícula regular de igual espaciado ∂, es decir, $x = i\partial$, $y = j\partial$, $z = l\partial$, y denotamos los puntos de la cuadrícula por $\mathbf{x}$.

Además, denotamos como $\mathcal{L}^2$ la aproximación PS de $\mathcal{L}^2$ en un muestreo de cuadrícula, de acuerdo con la ecuación 2. Por lo tanto, $\mathcal{L}^2$ es un operador de matriz discreta con las mismas dimensiones que la malla rectangular.

Es importante señalar que la DFT es una buena aproximación a la FT siempre que se cumpla el teorema de muestreo de Nyquist-Shannon en cada dimensión, por ejemplo, $k_{max} \leq \pi/\partial$, y la extensión periódica de la distribución espacial es continua. En este caso, el esquema PS tiene una precisión espacial de $O(\partial N)$.

7.1. Algoritmo de migración de tiempo inverso

El método numérico descrito anteriormente se ha utilizado para implementar el algoritmo de migración de tiempo inversión (RTM). RTM es un una migración de ecuación de onda acústica bidireccional preapilada para obtener imágenes precisas en y debajo de áreas con grandes complejidades estructurales y de velocidad constante/variable. RTM tiene un historial probado para generar imágenes de alta calidad y se utiliza cada vez más para refinar los límites estructurales de dominios computacionales durante la construcción de modelos de velocidad o de presión.

Para obtener imágenes de alta fidelidad, RTM calcula soluciones numéricas para la ecuación de onda completa (ver ecuación 1). Como tal, no tiene limitación de inmersión y maneja todas las formas de onda complejas de múltiples rutas. Históricamente, RTM se consideró poco práctico debido a los altos costos computacionales y una mayor sensibilidad a los parámetros de velocidad y reflectividad que los métodos continuos unidireccionales más establecidos. Sin embargo, la mejora constante de las arquitecturas computaciones, junto con flujos de trabajo de creación de modelos de velocidad/presión más precisos, han hecho viable el desarrollo e implementación de algoritmos RTM para la generación de imágenes en diferentes medios.

7.2. Estructura del código

El prototipo resuelve el problema de la propagación de una onda acústica a través de diferentes medios. Se ejecuta con un pequeño script llamado *"run"* que se encarga de lanzar tres ejecuciones, una para cada fase del programa. Estas tres fases se denominan: **preprocesado**, **kernel** y **postprocesado**.

7.2.1. Dependencias

Primeramente para poder ejecutar el prototipo es necesaria la preparación del entorno de desarrollo con la instalación de dependencias y software de terceros. A continuación se expondrán todas las herramientas utilizadas.

El sistema operativo utilizado es linux en concreto he escogido la distribución de Ubuntu en su versión 18.04 LTD. Una vez instalado el sistema operativo se requieren instalar las dependencias.

MPICH es una implementación portable, de alto rendimiento y open source de MPI, un estándar para el envío de mensajes para aplicaciones con memoria distribuida utilizada en la computación paralela.

HDF5 es un formato de almacenamiento de datos jerárquico pensado para organizar grandes cantidades de datos. Tiene integrado herramientas de alto rendimiento para optimizar tanto el tiempo de acceso a los datos como el espacio de almacenamiento. Además, también ofrece herramientas para consultar, manipular, visualizar y analizar los datos.

FFTW es conocida por ser la implementación open source más rapida de la *"Fast Fourier Transform (FFT)"*, un algoritmo eficiente que permite calcular la transformada de fourier discreta (DFT) en una o más dimensiones de tamaño de entrada arbitrarios.

A continuación se instaló Python en su versión 3.8.5, pero al tener problemas a la hora de instalar ciertos paquetes de Python necesarios finalmente se optó por la versión 3.6.9. Los paquetes de Python utilizados son los siguientes.

- **Numpy**: proporciona objetos de arrays multidimensionales conjunatamente con una amplia lista de rutinas muy eficientes para operar dichos arrays.

- **mpi4py**: nos permite utilizar MPI desde Python.
- **mpi4py-fft**: incluye una interfaz de FFTW para Python y junto a mpi4py nos permite distribuir arrays muy grandes gesationando las comunicaciones entre procesos
- **h5py**: es una interfaz de HDF5 para Python que nos permite almacenar grande cantidades de datos numéricos y facilidades para manipular los datos desde Numpy.
- **PyYAML**: YAML es un formato de serialización de datos diseñado para la legibilidad humana. PyYAML es un parser y emisor para Python de YAML.
- **Cython**: Es un lenguaje de programación para simplificar la escritura de módulos de extensión para Python en C y C++.
- **Scipy**: Es una biblioteca libre y de código abierto que se compone de herramientas y algoritmo matemáticos.
- **Sphinx**: Librería para generar documentación de Python.
- **singleton_decorator**: Permite crear objetos *"singleton"* a partir de un decorador.
- **colorama**: Permite colorear el texto en la terminal y el posicionamiento del cursor.
- **matplotlib**: Biblioteca para visualización gráfica en 2D y 3D.

Al momento de tener todas las dependencias necesarias para ejecutar el prototipo se estudió el funcionamiento de Git y GitHub para poder crear un repositorio en el que se pudiese compartir el prototipo y así en un futuro ir actualizándolo con las mejoras implementadas.

7.2.2. Entrada y salida

La entrada se compone del archivo *"params.yaml"* que contiene los parámetros mostrados en la Figura 3.

```
##############################################################################
# GENERAL PARAMETERS SECTION FOR IR TEST CASE
##############################################################################
# Preprocessing parameters
preprocessing:
  filename: data/IR_input_data2D.h5 # Input model file
  material_property: velocity      # Material name
  grid_meta_filename: grid_metadata.h5 # Out filename for grid metatada
  grid_data_filename: grid_data.h5    # Out filename for grid data
  receiver_meta_filename: receiver_metadata.h5 # Out filename for receiver metadata

# Execution parameters
run:
  ppw: 2          # Points per wavelength
  cfl: 0.95        #
  final_time: 4   #
  padding: 4000   #
  src_type: gaussian
  src_halo: 4     # Number of halo source points
  scale_strat: 0  # scaling strategy: 0: interpolate then correlate (accurate)
                  #                   1: correlate then interpolate at the end (not accurate)

# Boundary parameters
abc:
  sponge_mu: 0.02  #
  nodes: 30        # PML length (number of points)
  type: damped     # damped or sponge
# Output parameters
output:
  mode: image                   # Options supported: modelling, image
  debug_snapshots: 0.1          #
  debug_snapshots_time_skip: 2  #
  compression: 0                # 0: no compression, 1: compression by requantization
  compression_blockcount: 9    #  Amount of blocks used in compression in each direction
  directory: out                # Directory for output (results)
  directory_scratch: tmp        # Directory for temporal files
```

Figura 3: *params.yaml*

Además hay que tener un archivo HDF5 con los datos del modelo de velocidades, la fuente y el receptor. Por ello contamos con dos scripts escritos en Python *"build_IR_2Ddata.py"* y *"build_IR_3Ddata.py"* que se encargan de crear los archivos *"IR_input_data2D.h5"* y *"IR_input_data3D.h5"* respectivamente. Estos contienen un modelo de velocidad homogéneo, es decir, la velocidad es constante para todos los puntos del modelo y por lo tanto afectarán de la misma forma a la onda.

La salida de la aplicación es un archivo HDF5 con los siguientes datos:

- **axis_order**: Orden de los ejes (ZX/ZXY).
- **code_version**: Versión del código.
- **date**: La fecha de la ejecución.
- **machine**: Nombre de la máquina en la que se realizó la ejecución.
- **dh**: Variación en el espacio (Delta in space).

- **dt**: Variación en el tiempo (Delta in time).
- **nt**: Número de pasos temporales.
- **num_dimensions**: Número de dimensiones (2D/3D).
- **num_grid_points**: Número de puntos de la cuadrícula.
- **num_processors**: Número de procesos con los que se ha ejecutado el programa.
- **image**: Valores de la imagen de la propagación de la onda captada por el receptor.

7.2.3. Preprocesado

El preprocesado es totalmente secuencial y está compuesto de las siguientes partes.

1. **Lectura de la entrada**: Consiste en parsear el archivo de entrada y crear objetos con todos los datos del archivo.
2. **Inicialización**: Esta parte se encarga inicializar el conjunto de datos del modelo, las fuentes y los receptores haciendo lecturas del archivo hdf5 especificado en la entrada. Además, se crea el directorio temporal y el directorio donde se alojará el archivo de salida.
3. **Calcular la cuadrícula espacio-tiempo**: En esta parte se calcula la cuadrícula espacio-temporal que es una discretización del modelo de entrada y por lo tanto se interpolará el modelo a dicha cuadrícula. Esta discretización está adaptada al proceso, es decir que el tamaño de la cuadrícula dependerá del modelo, la frecuencia máxima de la onda y la variable de entrada de ppw (points per wavelength). En la Figura 4 se muestra un ejemplo. Se guardan tanto los datos como los metadatos calculados en archivos temporales.
4. **Calcular metadatos del receptor**: Consiste en computar los datos necesarios para el receptor y en guardarlos en archivos temporales.

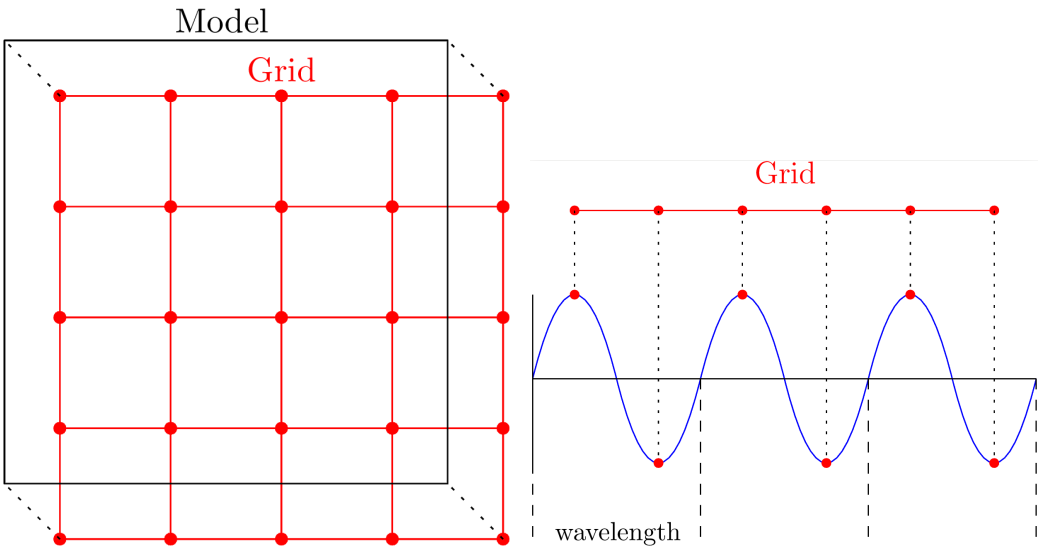


Figura 4: *Ejemplo de cuadrícula espacio-tiempo para un modelo 2D (izquierda) y boceto de cuadrícula con ppw=2 (derecha). Elaboración propia*

7.2.4. Kernel

El kernel es la única parte del prototipo que se hace en paralelo y se compone de las siguentes partes.

1. **Lectura de la entrada**: Es exactamente la misma parte que en el preprocesado.
2. **Inicialización**: Lo único que lo diferencia de la fase del preprocesado es que se inicializan los timers.
3. **Importar metadatos de la cuadrícula espacio-tiempo**: Se cargan los metadatos de la cuadrícula espacio-temporal calculada en la fase del preprocesado leyendo el archivo.
4. **Importar metadatos del receptor**: Se cargan los datos del receptor calculada en la fase del preprocesado leyendo el archivo.
5. **Crear estructuras paralelas**: Creación de las estructuras de datos necesarias para paralelizar la aplicación y lectura de los datos de la cuadrícula.

6. **Crear estructuras de la fuente y del receptor**: Creación de las estructuras de la fuente y calcular la descomposición del dominio del receptor

7. **Crear estructuras de los límites**: Consiste en inicializar los datos de los límites.

8. **Inicializar y ejecutar solver**: Inicializa los datos del Solver y lo ejecuta. El Solver es el que se encarga de resolver el sistema de ecuaciones y de toda la parte matemática del problema. Cuenta con dos fases, *"forward"* donde se simula hacia delante en el tiempo (0..nt) y *"backward"* donde se simula hacia atrás en el tiempo (nt..0).

7.2.5. Postprocesado

Por último la fase del postprocesado que también es completamente secuencial y está compuesto de las siguientes partes.

1. **Lectura de la entrada**: Es exactamente la misma parte que en el preprocesado.

2. **Inicialización**: Es exactamente la misma parte que en el preprocesado.

3. **Importar metadatos de la cuadrícula espacio-tiempo**: Es exactamente la misma parte que en el kernel.

4. **Postprocesar Solución**: Lectura de los datos calculados en el Solver, interpolación de la cuadrícula al modelo y escritura de los datos de salida.

El prototipo cuenta con dos modos de ejecución *"modelling"* e *"image"*. El modo *"modelling"* genera un fichero que puede ser dado como entrada al modo *"image"* y tan solo ejecuta la fase forward del Solver. Sin embargo en el modo *"image"* se ejecutan las dos fases.

7.3. Simulaciones y resultados

7.3.1. Test de validación: modelo impulso-respuesta

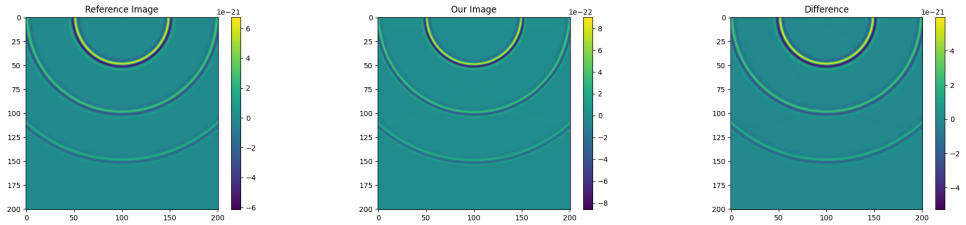
Para verificar la implementación del algoritmo RTM, se una utilizado un modelo de impulso-respuesta (I-R). Una función de I-R describe la evolución de la variable de interés a lo largo de un horizonte de tiempo especificado después de un choque de onda en un momento dado. En otras palabras, una función I-R define cómo un sistema/modelo responde a alguna señal de entrada o impulso. Una descripción amplia y detallada se puede encontrar en *Fletcher, R. (2009)[11].*

Este test lo utilizaremos para verificar la precisión numérica y performance (escalabilidad y tiempos). El modelo I-R utilizado cuenta con tres impulsos en tiempo (tres picos). Los detalles de cada configuración y experimento se detallan en secciones más adelante.

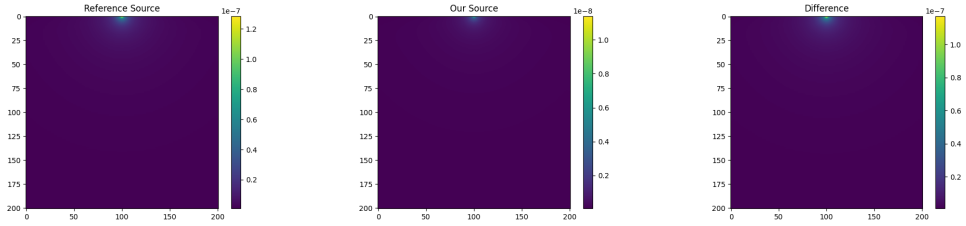
7.3.2. Precisión numérica

Para verificar que los resultados en secuencial son correctos contamos con tres archivos binarios que provisionan de la imagen, la fuente y el modelo de referencia de la salida del caso base homogéneo 2D. Además contamos con un script para visualizar tanto nuestra salida, como la de referencia y sus diferencias.

(a) Imagen de referencia (b) Imagen PSM (c) Diferencia (imagen)



(d) Fuente de referencia (e) Fuente PSM (f) Diferencia (fuente)



(g) Solución de referencia (h) Solución PSM (i) Diferencia (solución)

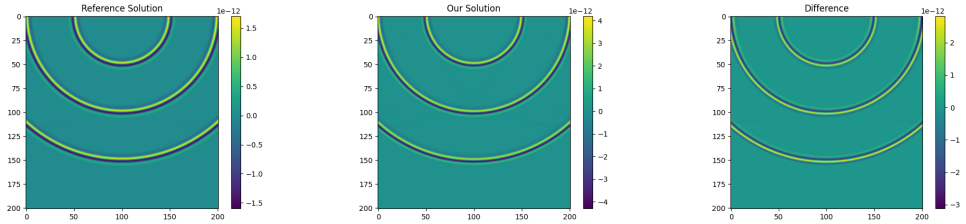


Figura 5: *Visualización de la precisión numérica. Elaboración propia.*

Como se puede ver en la Figura 5 en este caso en concreto la precisión que cuenta el algoritmo para la imagen es de aproximadamente $\pm 6e^{-21}$, para la fuente de $+1{,}2e^{-7}$ y para la solución $\pm 3e^{-12}$

7.3.3. Tiempos y consumo de memoria en un ordenador personal

Se han realizado tests en el ordenador personal del programador capturado el tiempo de ejecución y el consumo de memoria para ejecuciones con número de threads desde 1 hasta 5 haciendo la media de cuatro ejecuciones. Se han utilizado los casos base de modelos homogéneos de tamaño 201x201 y 201x201x201 para 2D y 3D respectivamente. Los resultados de estos experimentos se presentan en la Figura 6.

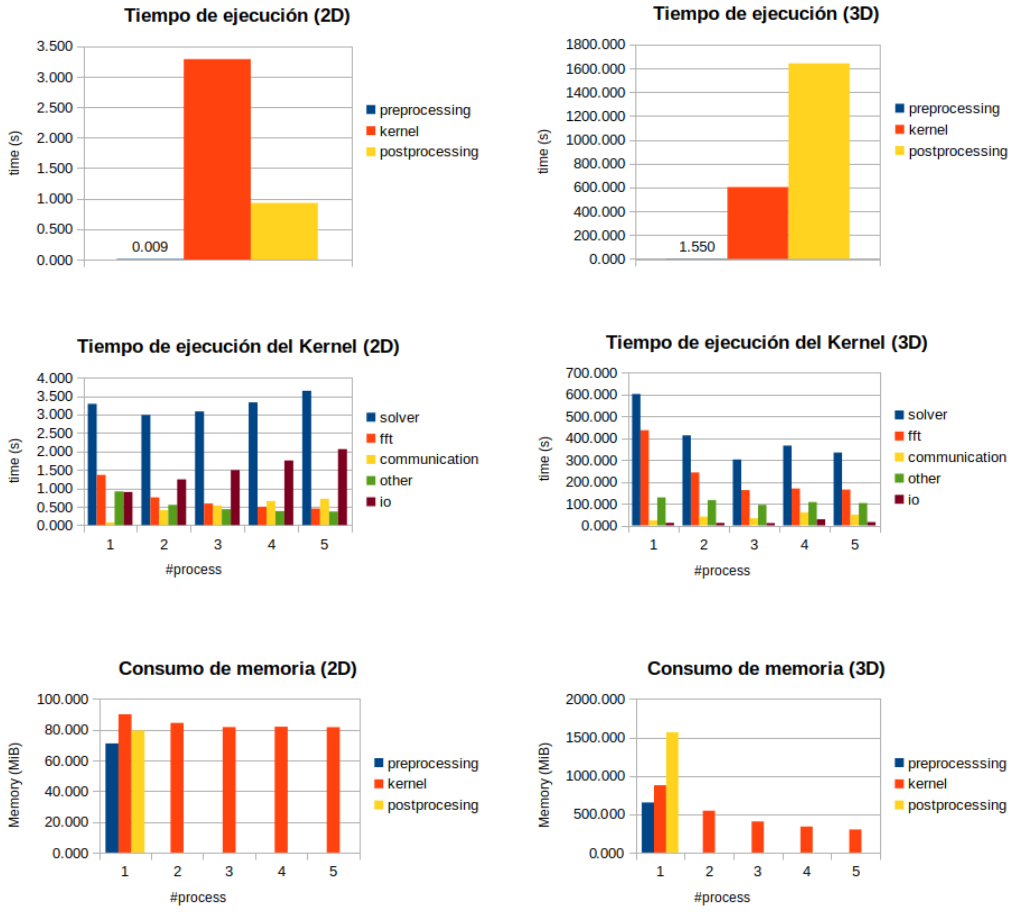


Figura 6: *Test de tiempo de ejecución y consumo de memoria del prototipo.*

Como se puede observar, el postprocesado es la parte que peor escala respecto al tamaño de la entrada, tanto para el tiempo de ejecución como para el consumo de memoria.

7.3.4. Análisis de escalabilidad

Se han realizado test del tiempo de ejecución en MN. El sistema cuenta 48 racks con 3456 nodos. Estos test se han realizado para el modo *"modelling"* e *"image"* utilizando 1 y 2 nodos. Cada nodo está equipado con:

- 2 sockets Intel Xeon Platinum 8160 CPU con 24 cores cada uno a @2.10GHz con un total de **48 cores por nodo**.
- L1d 32K; L1i cache 32K; L2 cache 1024K; L3 cache 33792K.
- 96 GB de memoria principal **1.880GB/core**, 12x 8GB 2667Mhz DIMM (216 nodos con gran memoria, 10368 cores con 7.928GB/core).
- 100 Gbit/s Intel Omni-Path HFI Silicon 100 Series adaptador PCI-E.
- 10 Gbit Ethernet.
- 200 GB SSD local accessible como espacio de almacenamiento temporal durante las tareas.

Simulaciones 2D

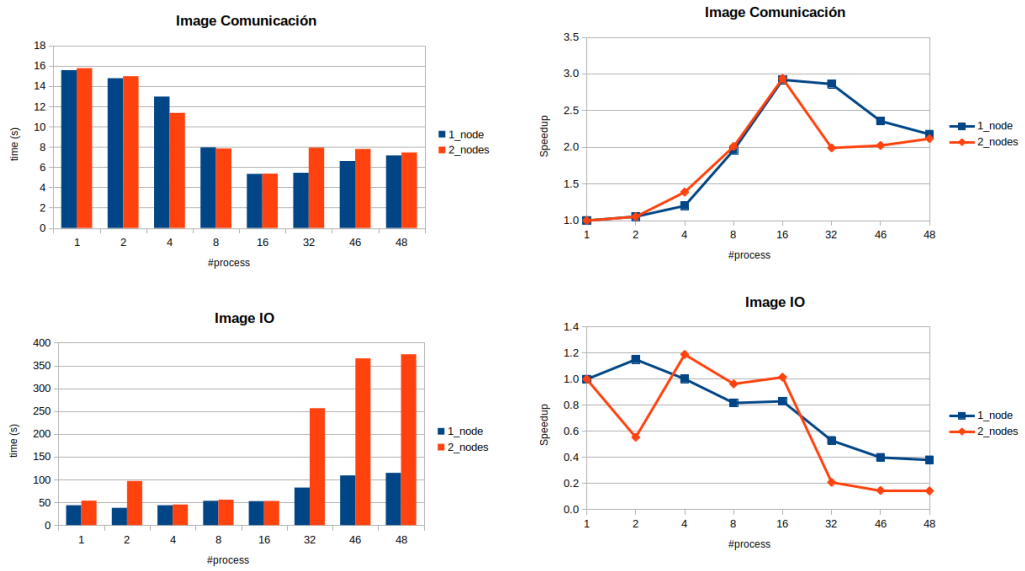


Figura 7: *Observable 1 (2D). Elaboración propia.*

En la Figura 7 podemos observar que ejecutando con el modo image, donde la aplicación escala peor es en la comunicación y en la entrada y salida (IO). Por lo que hace la comunicación escala peor sobre todo cuando se ejecuta en 2 nodos debido a que la comunicación entre nodos es más lenta que la comunicación intra nodo. También hay que tener en cuenta que los procesos no se distribuyen inteligentemente entre sockets. [1]

[1]Dirigirse al Anexo A para consultar todas las gráficas tanto para los casos 2D como los 3D [9].

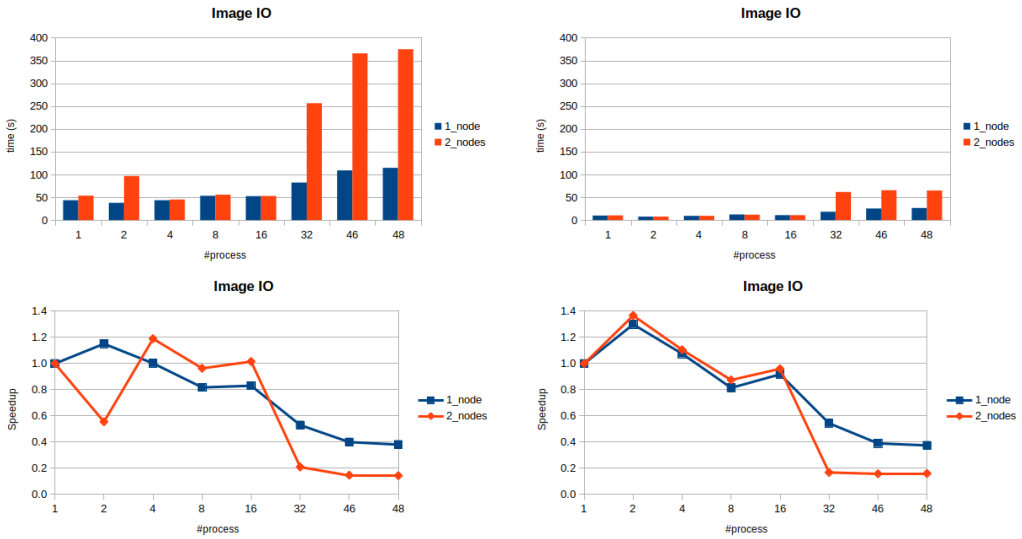


Figura 8: *Observable 2 (2D). debug_snapshot_time_skip = 2 (izquierda) y debug_snapshot_time_skip = 6 (derecha). Elaboración propia.*

Por lo que hace a la entrada y salida (IO), y por lo que podemos observar en la Figura 8, tiene muy mala escalabilidad. Esto es debido al sistema de colas de los ficheros de MN. Para gestionar este aspecto, el prototipo cuenta con el parámetro de entrada llamado *"debug_snapshot_time_skip"* que indica cada cuántas iteraciones de tiempo guardamos los datos en disco. Se puede observar una gran mejora en el tiempo de ejecución con el valor de esta variable a 6 respecto con el valor a 2. Pero la IO es un problema que no se ha contemplado en este trabajo que quedará para el trabajo a futuro.

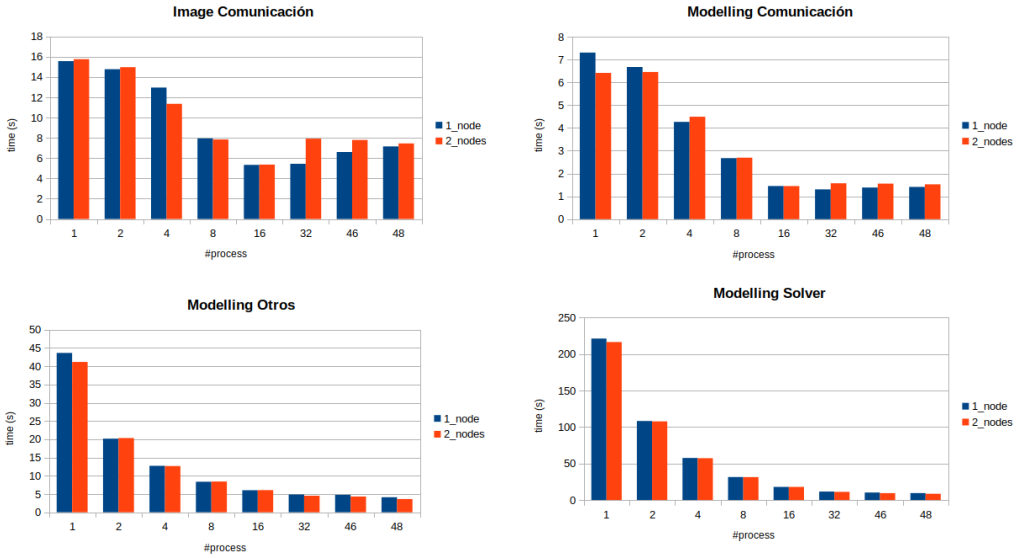


Figura 9: *Observable 3 (2D). Elaboración propia.*

Sin embargo, ejecutando con el modo *"modelling"* (Figura 9), la comunicación se ve reducida a la mitad respecto al modo *"image"* ya que solo se ejecuta la fase forward y por lo tanto tiene la mitad de fases. Pero para la fase otros, que son operaciones matriciales, a medida que se aumentan los cores acaba ocupando aproximadamente la mitad del tiempo del todo el Solver.

Otra punto a discutir es que sabiendo el tiempo de ejecución del solver y su número de iteraciones para una malla de dimensiones $n * m$ es posible calcular el tiempo de una iteración. Esto es útil para aproximar el tiempo que tardará otra ejecución con un número de iteraciones distinto pero con la misma malla. Lo ideal es calcular este número para mallas de tamaño representativo que lo dejaremos para el trabajo a futuro.

Simulaciones 3D

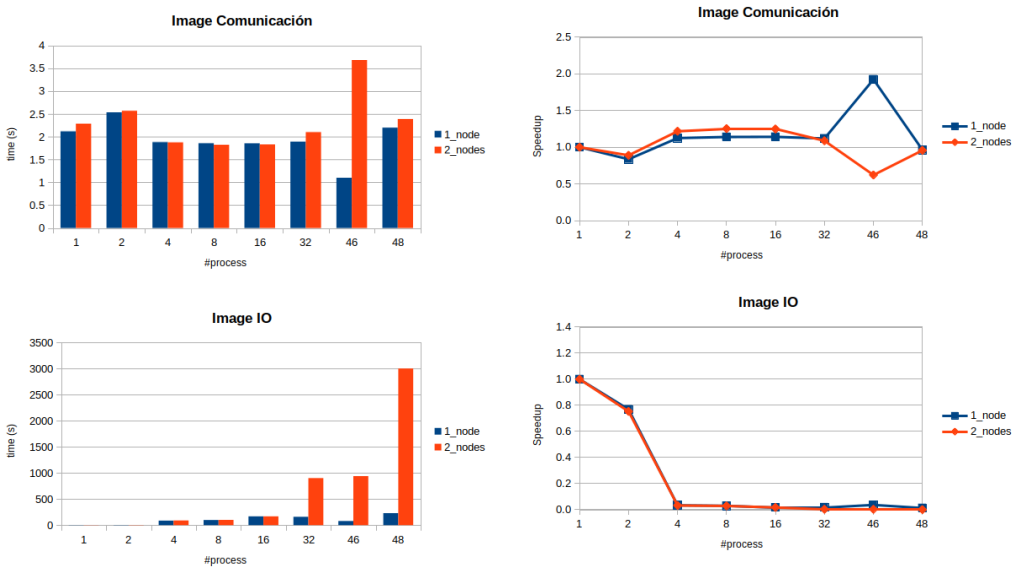


Figura 10: *Observable 1 (3D). Elaboración propia.*

Por lo que hace a las simulaciones 3D podemos observar los mismos resultados. En la Figura 10 vemos que con el modo *"image"* las fases que peor escalan son la comunicación y la IO por las razones comentadas anteriormente.

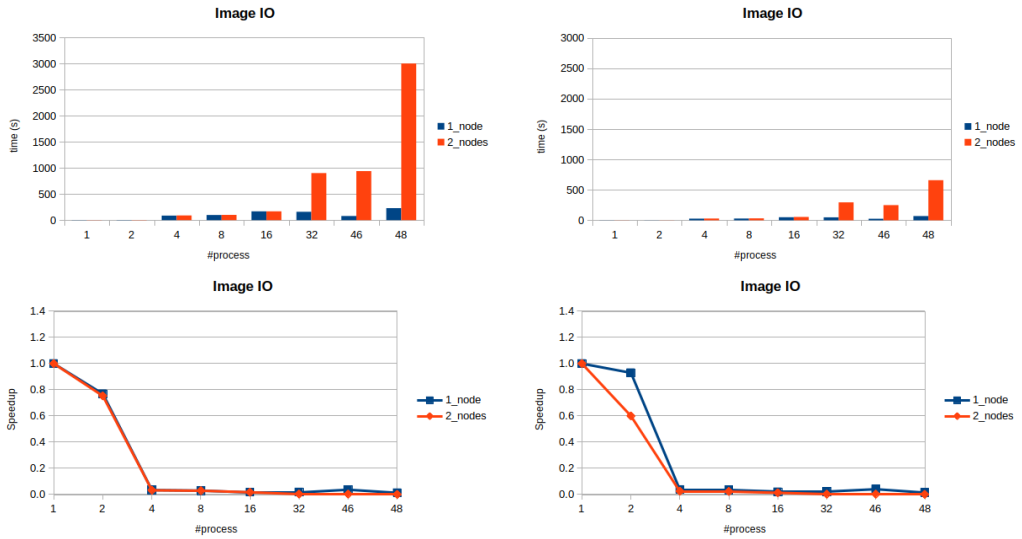


Figura 11: *Observable 2 (3D). debug_snapshot_time_skip = 2 (izquierda) y debug_snapshot_time_skip = 6 (derecha). Elaboración propia.*

En la Figura 11, la IO se comporta como se esperaba a la hora de cambiar el valor del parámetro *"debug_snapshot_time_skip"*.

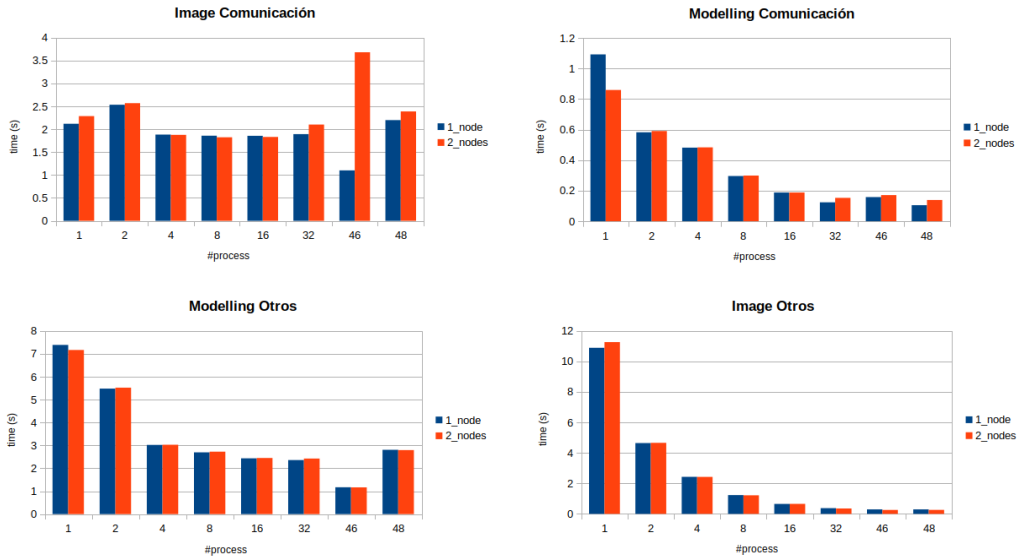


Figura 12: *Observable 3 (3D). Elaboración propia.*

Sin embargo, en la Figura 12, a diferencia de las simulaciones 2D esta vez las comunicaciones se mantienen practicamente constantes para el modo image. Y para el modo *"modelling"* la fase otros tiene mucho peor rendimiento para casos grandes como es la simulación 3D.

8. Implementación de mejoras

8.1. Docker

Al contar con incompatibilidades con algunas versiones de algunas de las dependencias mencionados anteriormente se tomó la decisión de utilizar docker para remediarlo. Docker permite a los usuarios crear aisladamente e independientemente entornos para ejecutar y lanzar sus aplicaciones. Estos entornos se llaman contenedores. Con esto podemos crear un contenedor con todo el software necesario y con sus versiones compatibles para así poder ejecutar nuestra aplicación en cualquier máquina que disponga de docker.

Para crear este contenedor es necesario crear una imagen de docker a partir de un archivo creado por nosotros llamado *"Dockerfile"*. Con este archivo podemos utilizar una imagen creada por docker de Ubuntu 18.04 LTD e instalar todas las dependencias. Una vez hecha la imagen es posible ejecutarla y hacer uso de nuestra aplicación.

8.2. Flujo de trabajo del código

El primer cambio consistió en actualizar las funciones de medición de tiempos. Más concretamente, se han implementado timers de MPI en lugar de timers de funciones nativas de Python. Esto permite obtener mediciones de tiempo más precisas en ejecuciones paralelas.

A partir de observar qué hay partes repetidas entre las tres fases había que integrarlas, manteniendo así en memoria principal los datos y objetos que necesitamos permitiéndonos eliminar lecturas y escrituras para el paso de datos y metadatos entre las fases.

Una vez integradas las fases, el siguiente paso fue paralelizar tanto la fase del preprocesado como la fase del postprocesado para aprovechar al máximo el paralelismo. Para ello se necesita paralelizar las interpolaciones de las dos fases distribuyendo así el modelo y la cuadricula entre los cores. Gracias a la librería de *"mpi4py_fft"* podemos utilizar su módulo PFFT para distribuir las matrices de dos formas diferentes, Slab decomposition donde tan solo una dimensión está distribuida y Pencil decomposition donde dos dimensiones están distibuidas.

Con esta librería estamos obligados a tener como mínimo un eje alineado lo que implica que para 2D tan solo podemos utilizar la Slab decomposition. En las Figuras 13 y 14 se muestra un ejemplo de cada descomposición.

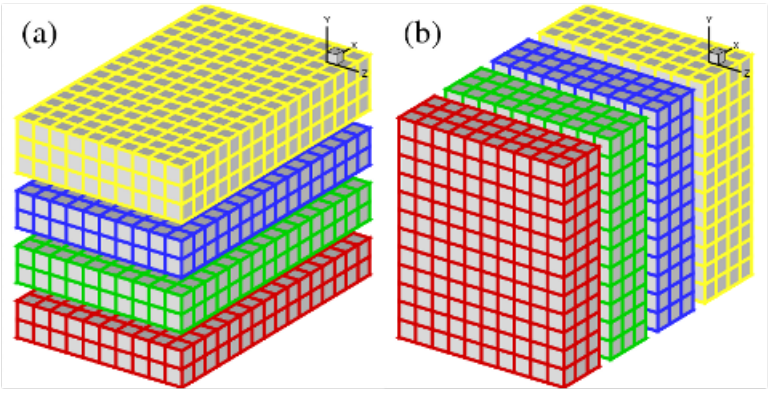


Figura 13: *Ejemplo de slab decomposition con 4 cores.*
(a) descomposición en el eje Y; (b) descomposición en el eje X.
Tomado de la página de 2decomp&FFT[12].

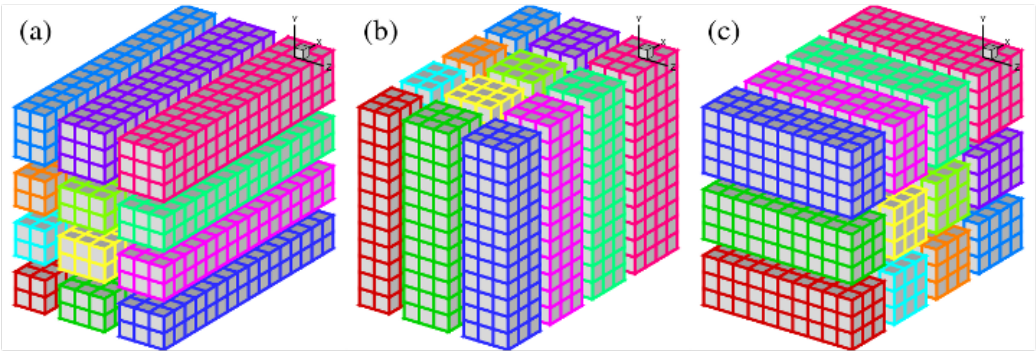


Figura 14: *Ejemplo de pencil decomposition con 4*3 cores.*
(a) X-pencil; (b) Y-pencil; (c) Z-pencil.
Tomado de la página de 2decomp&FFT[12].

Si solo contamos con modelos de velocidad homogéneos no se requiere ninguna implementación adicional. Sin embargo, este no es el caso del problema que se pretende resolver en este TFG. El programa debe aceptar escenarios realistas y complejos, por lo que supone que los modelos de velocidad serán en su mayoría heterogéneos. Para conseguir una interpolación en paralelo correcta y suave de un modelo de dichas características, es necesario que cada proceso se comunique con sus vecinos para obtener la serie de datos que requieren y con ellos poder computar correctamente los valores de la frontera de cada proceso. Para esto se añadió un parámetro *interpolation: halo* que indica la anchura de puntos consultados a cada vecino. Ejemplo en la Figura 15.

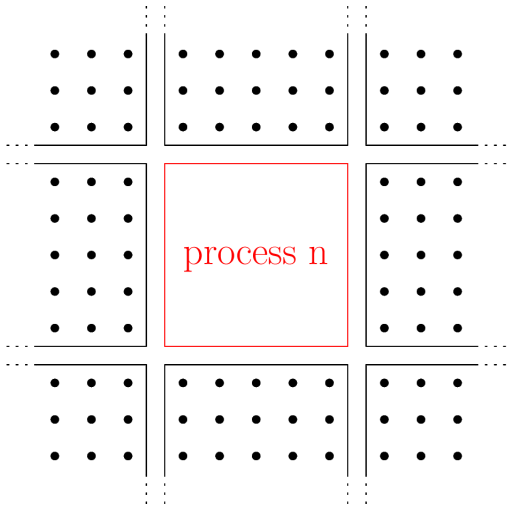


Figura 15: *Ejemplo de comunicación para la interpolación con halo* = 3. *Elaboración propia.*

Para comprobar que la implementación de la interpolación es correcta se han implementado test unitarios que nos permiten cambiar el tamaño de la entrada y de la salida además de sus valores homogéneos o heterogéneos(randoms).

Pero esta implementación nos acaba dando problemas a la hora de utilizarla con muchos cores y un gran consumo de memoria, ya que las funciones para interpolar que utilizamos de la librería Scipy consumen bastante memoria. Por lo que acabamos implementando una alternativa pero esta vez en secuencial.

La idea es dividir la matriz de valores a interpolar en partes pequeñas teniendo en cuenta que necesitamos el halo de valores de las partes vecinas y iterar sobre ellas realizando interpolaciones pequeñas y acabar haciendo una reducción para construir el resultado. El tamaño de cada iteración se ve definido por otro parámetro introducido llamado *interpolation: slice_width*.

9. Conclusiones y trabajo a futuro

En este TFG se han estudiado e implementado mejoras a un prototipo PSM para la simulación de ondas acústicas en 2D/3D. Se han analizando y desarrollado diferentes estrategias computacionales impactando positivamente en la eficiencia del método numérico y en la flexibilidad del código. Todo lo anterior bajo un enfoque open source.

La conclusión principal es que los objetivos se han alcanzado satisfactoriamente. Se ha demostrado que para cada una de las fases del algoritmo se ofrece un rendimiento distinto y se han implementado mejoras sustanciales en el código. Además, se ha creado un repositorio con soporte para Dockers que resultará de utilidad para futuros desarrollos a partir de la versión implementada en este TFG.

A nivel personal me ha aportado nuevos conocimientos técnicos y a su vez me ha permitido aportar y aplicar al proyecto todo el conocimiento adquirido durante los años de este grado. También me ha aportado nuevos conocimientos para poder desarrollar exitosamente toda la parte de gestión de un proyecto. No deja de ser cierto que, aunque la experiencia del proyecto ha sido enriquecedora y que la pandemia del SARS-CoV 2 no lo ha permitido, me hubiese gustado vivir la experiencia de la forma usual.

Como hemos comentado anteriormente, el trabajo a futuro consiste en mejorar la tarea de IO del algoritmo ya que el algoritmo FWI presentado en este documento requiere escrituras a disco eficientes. Para ello se podría implementar y analizar escrituras con un formato diferente como podria ser en binario, o buscar métodos para escribir menos bytes e incluso utilizar un sistema de compresión de datos. También se podrían utilizar nuevas tecnologías de memoria como Optane o aprovechar escrituras en memoria virtual.

A parte de esto, sería bueno extrapolar el tiempo por iteración para aproximar el tiempo total con varias mallas de tamaño representativo. También cambiar el repositorio de privado a público una vez se crea oportuno que esté listo para ser open source.

Referencias

[1] Geosciences Applications Group of the Barcelona Supercomputing Center.

https://www.bsc.es/es/discover-bsc/organisation/scientific-structure/geophysical-applications

[2] Octavio Castillo Reyes.

https://futur.upc.edu/OctavioCastilloReyes

[3] Google Meet.

https://meet.google.com/

[4] Git.

https://git-scm.com/

[5] GitHub.

https://github.com/

[6] Docker.

https://www.docker.com/

[7] Ubuntu.

https://ubuntu.com/

[8] Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública.

https://www.boe.es/boe/dias/2018/03/06/pdfs/BOE-A-2018-3156.pdf

[9] Tabla de coeficientes de amortización lineal.

https://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresas/Impuesto_sobre_Sociedades/Periodos_impositivos_a_partir_de_1_1_2015/Base_imponible/Amortizacion/Tabla_de_coeficientes_de_amortizacion_lineal_.shtml

[10] Normativa TFG GEI FIB UPC.

https://www.fib.upc.edu/sites/fib/files/documents/estudis/normativa-tfg-mencio-addicional-gei-br.pdf

[11] Fletcher, R. P., Du, X., & Fowler, P. J. (2009). Reverse time migration in tilted transversely isotropic (TTI) media. Geophysics, 74(6), WCA179-WCA187.

[12] Library for 2D pencil decomposition and distributed Fast Fourier Transform.

`http://www.2decomp.org/decomp.html`

Anexo

Anexo A: Test Escalabilidad

Este anexo contiene todas las gráficas obtenidas a partir de pruebas de escalabilidad ejecutadas en MN. Consultalas en las siguientes páginas $\downarrow\downarrow\downarrow$.

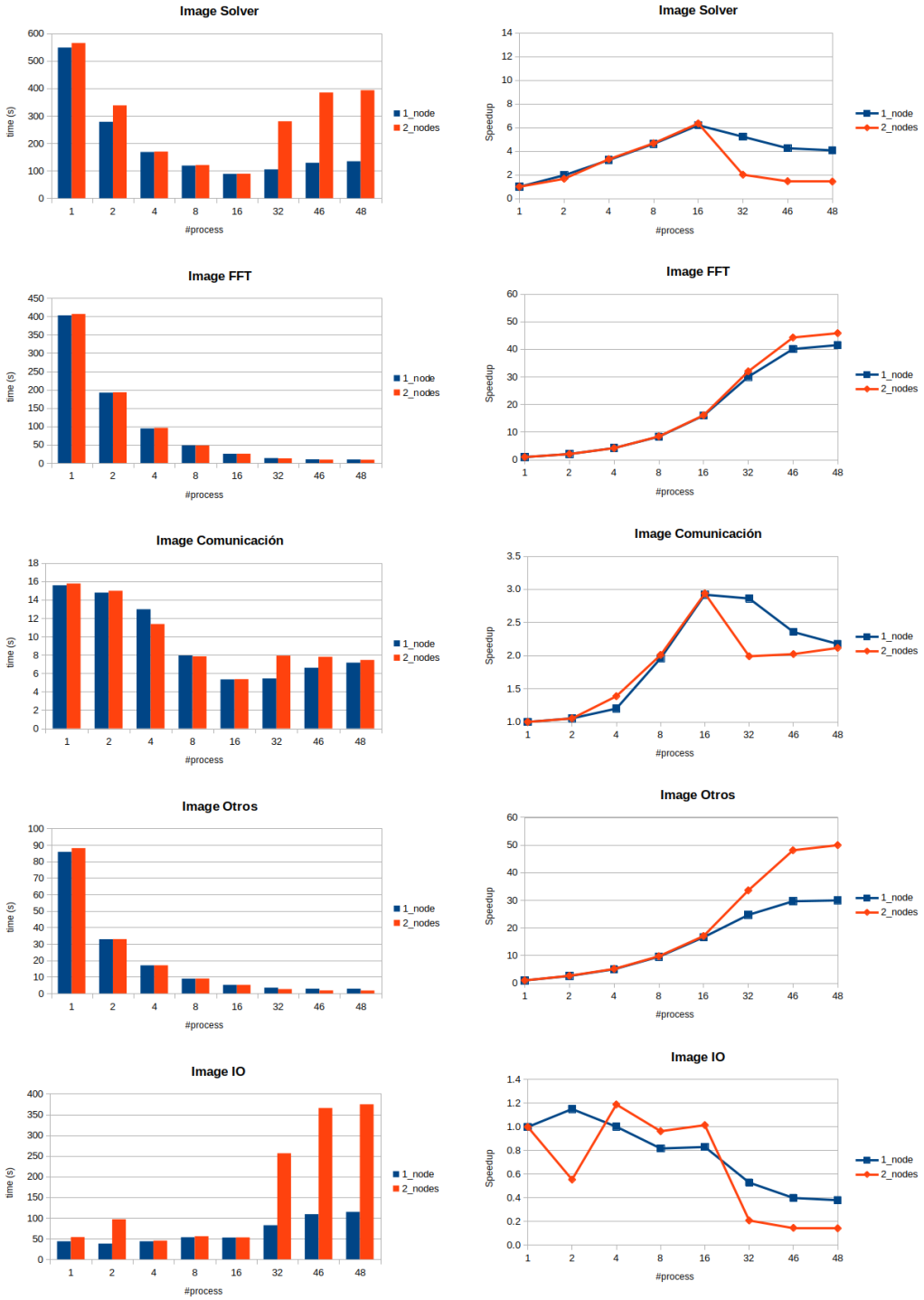


Figura 16: *Gráficas del tiempo de ejecución (izquierda) y speedup (derecha) del Solver 2D con el modo "image" con debug_snapshot_time_skip = 2. Elaboración propia.*

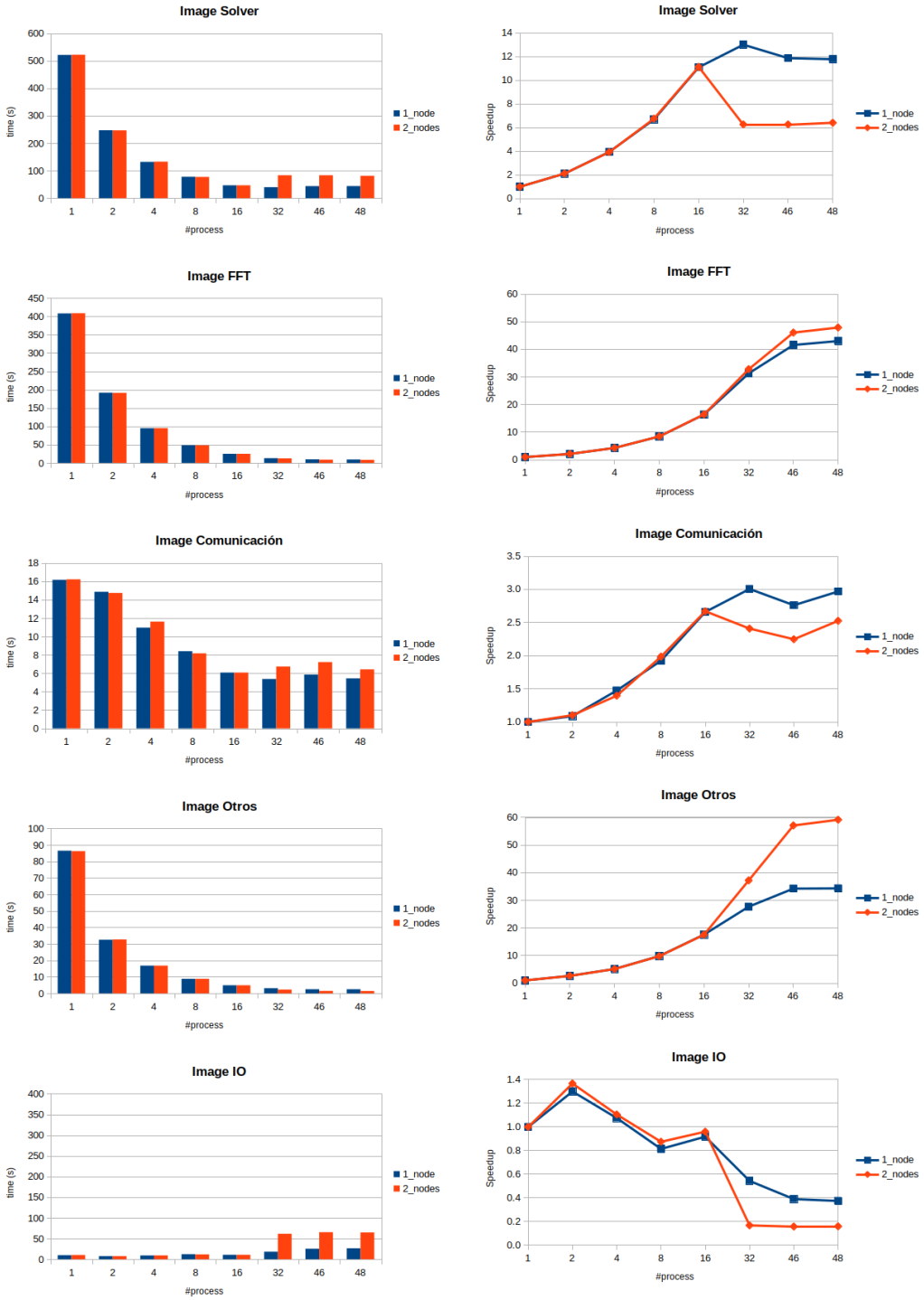


Figura 17: *Gráficas del tiempo de ejecución (izquierda) y speedup (derecha) del Solver 2D con el modo "image" con debug_snapshot_time_skip = 6. Elaboración propia.*

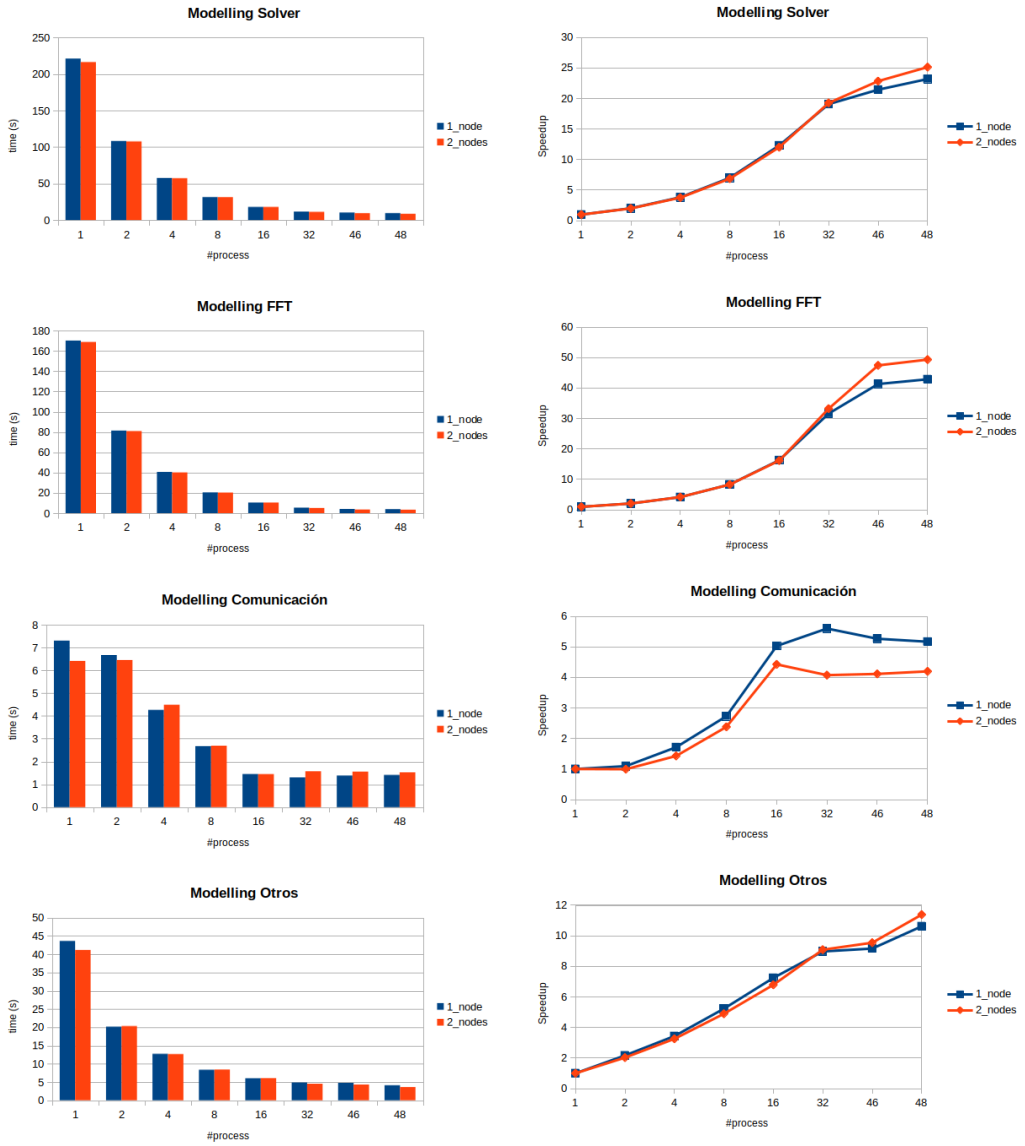


Figura 18: *Gráficas del tiempo de ejecución (izquierda) y speedup (derecha) del Solver 2D con el modo "modelling" con debug_snapshot_time_skip = 2. Elaboración propia.*

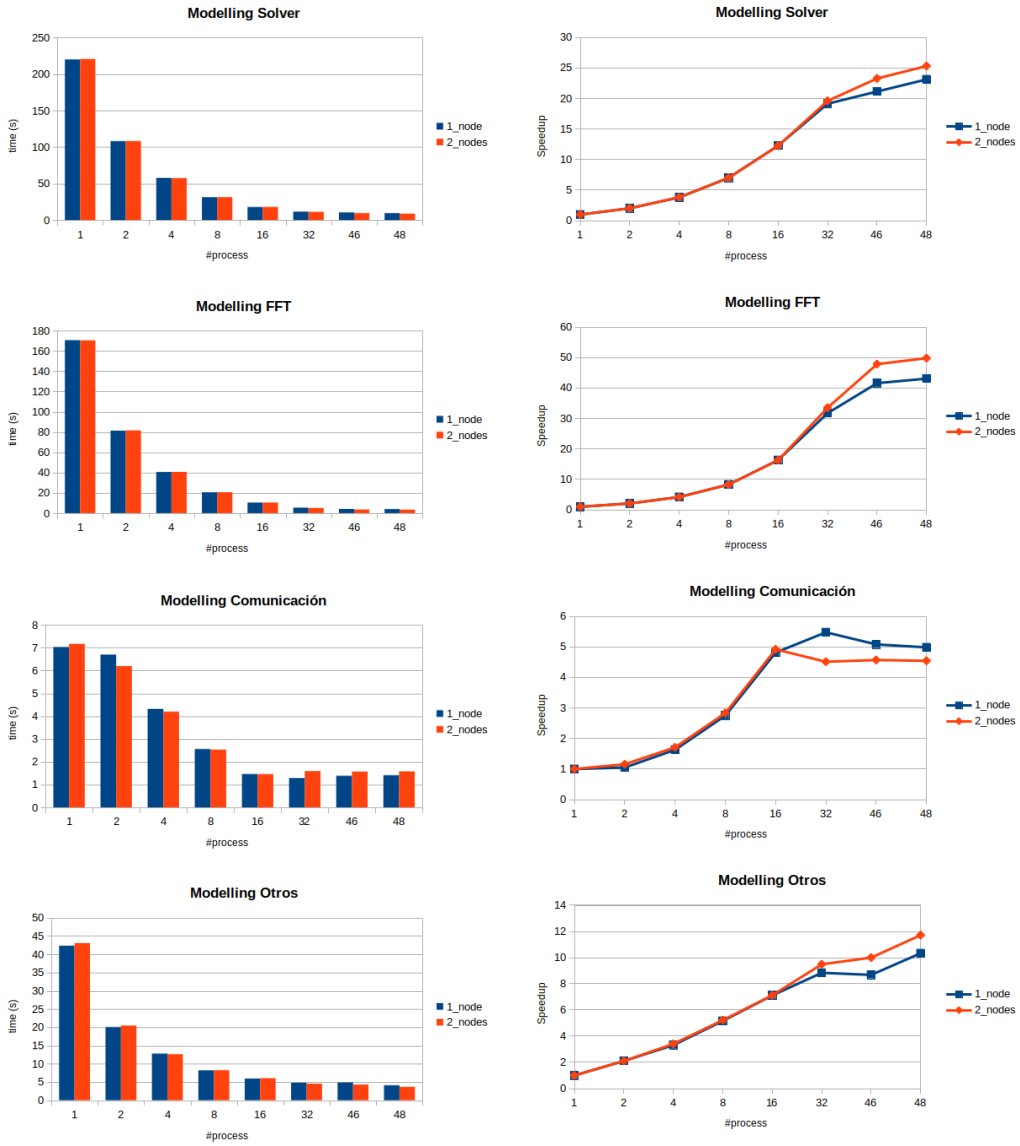


Figura 19: *Gráficas del tiempo de ejecución (izquierda) y speedup (derecha) del Solver 2D con el modo "modelling" con debug_snapshot_time_skip = 6. Elaboración propia.*

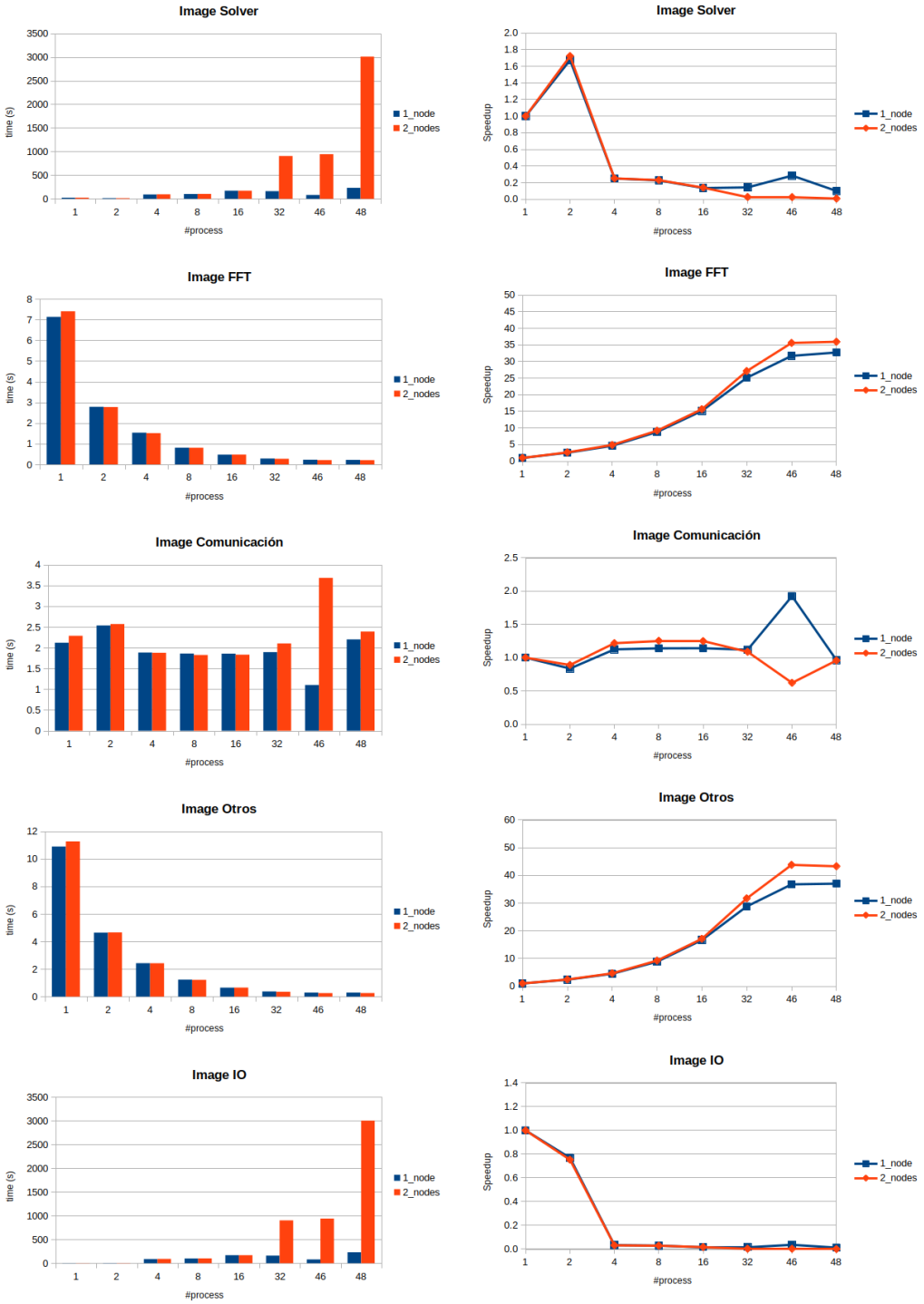


Figura 20: *Gráficas del tiempo de ejecución (izquierda) y speedup (derecha) del Solver 3D con el modo "image" con debug_snapshot_time_skip = 2. Elaboración propia.*

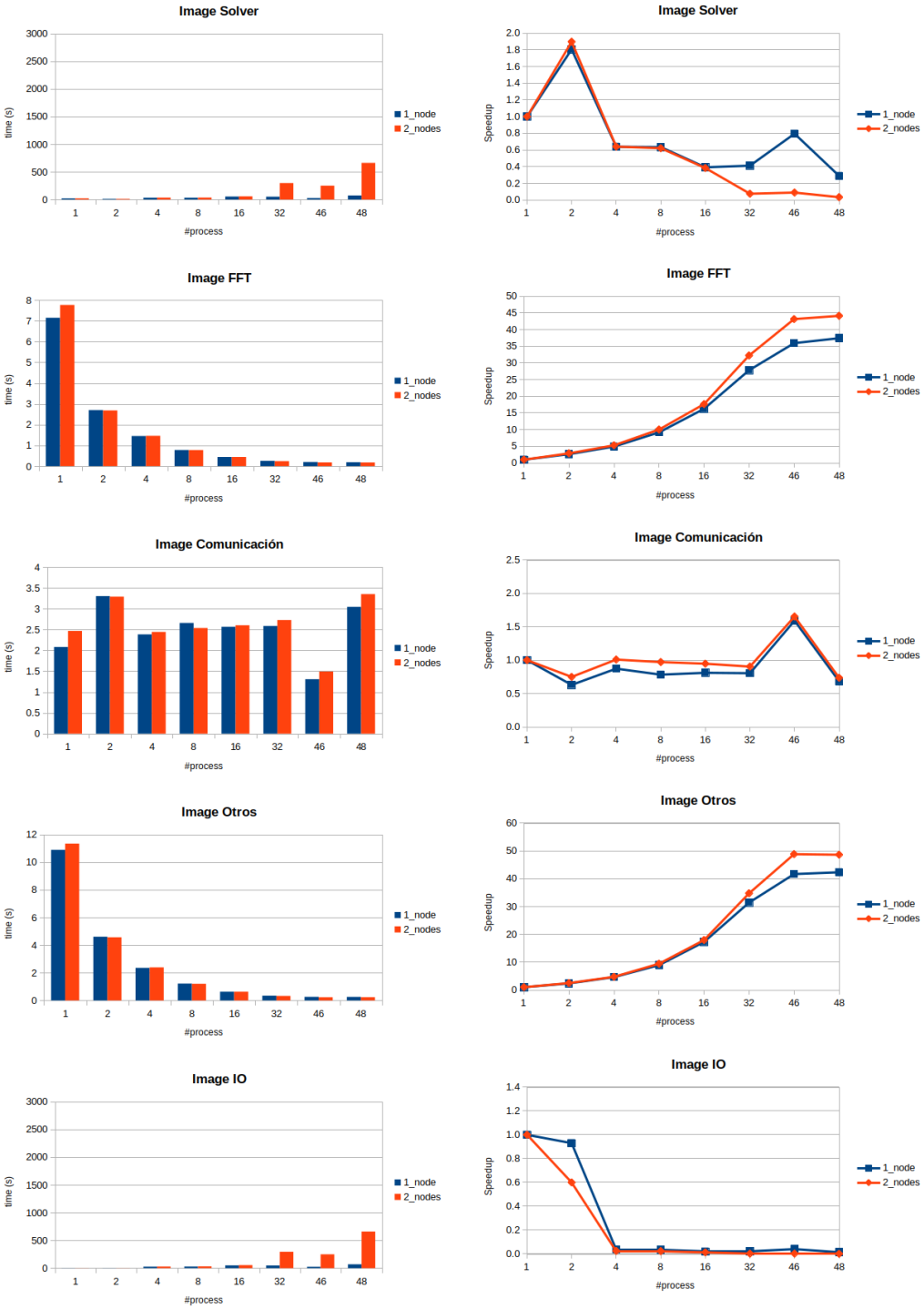


Figura 21: *Gráficas del tiempo de ejecución (izquierda) y speedup (derecha) del Solver 3D con el modo "image" con debug_snapshot_time_skip = 6. Elaboración propia.*

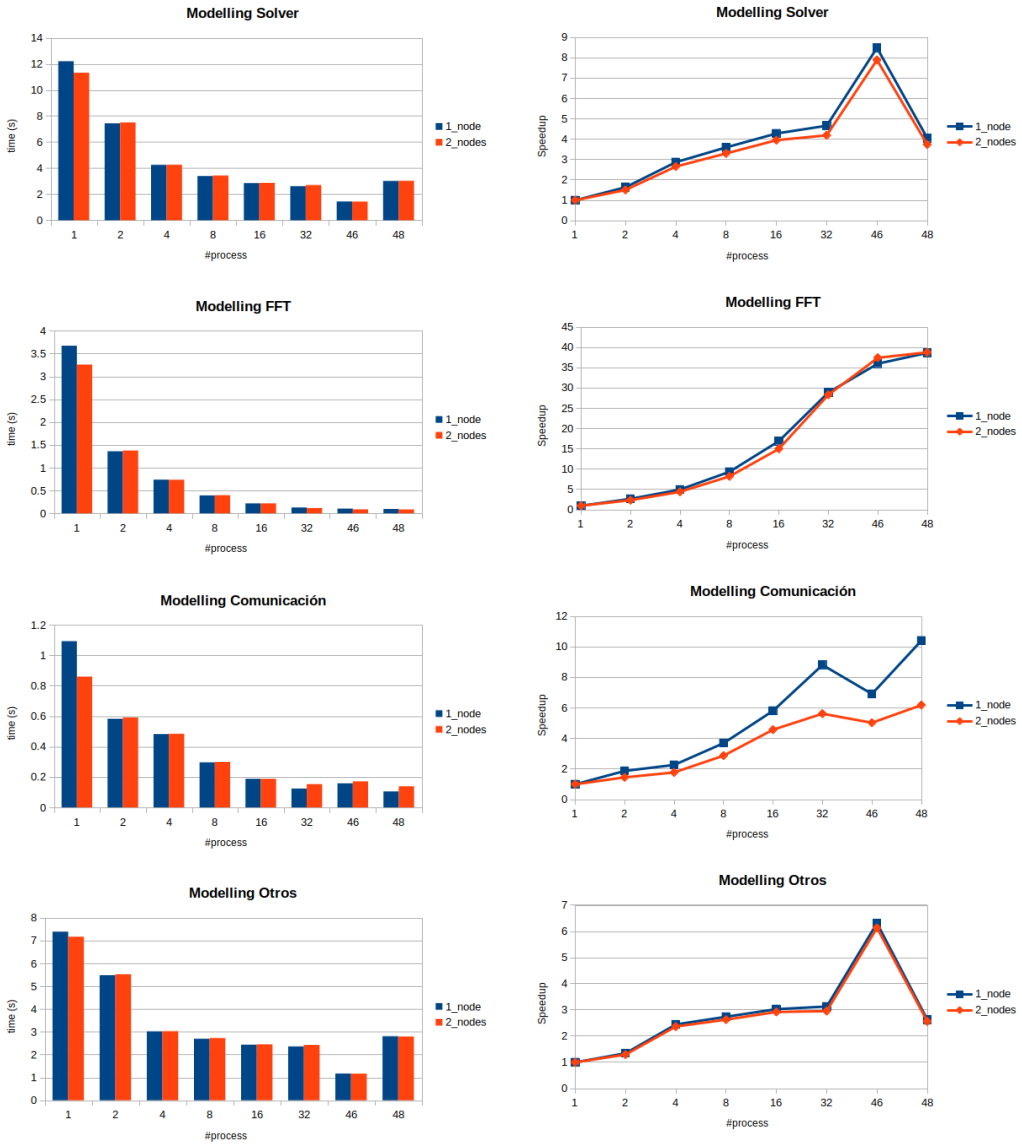


Figura 22: *Gráficas del tiempo de ejecución (izquierda) y speedup (derecha) del Solver 3D con el modo "modelling" con debug_snapshot_time_skip = 2. Elaboración propia.*

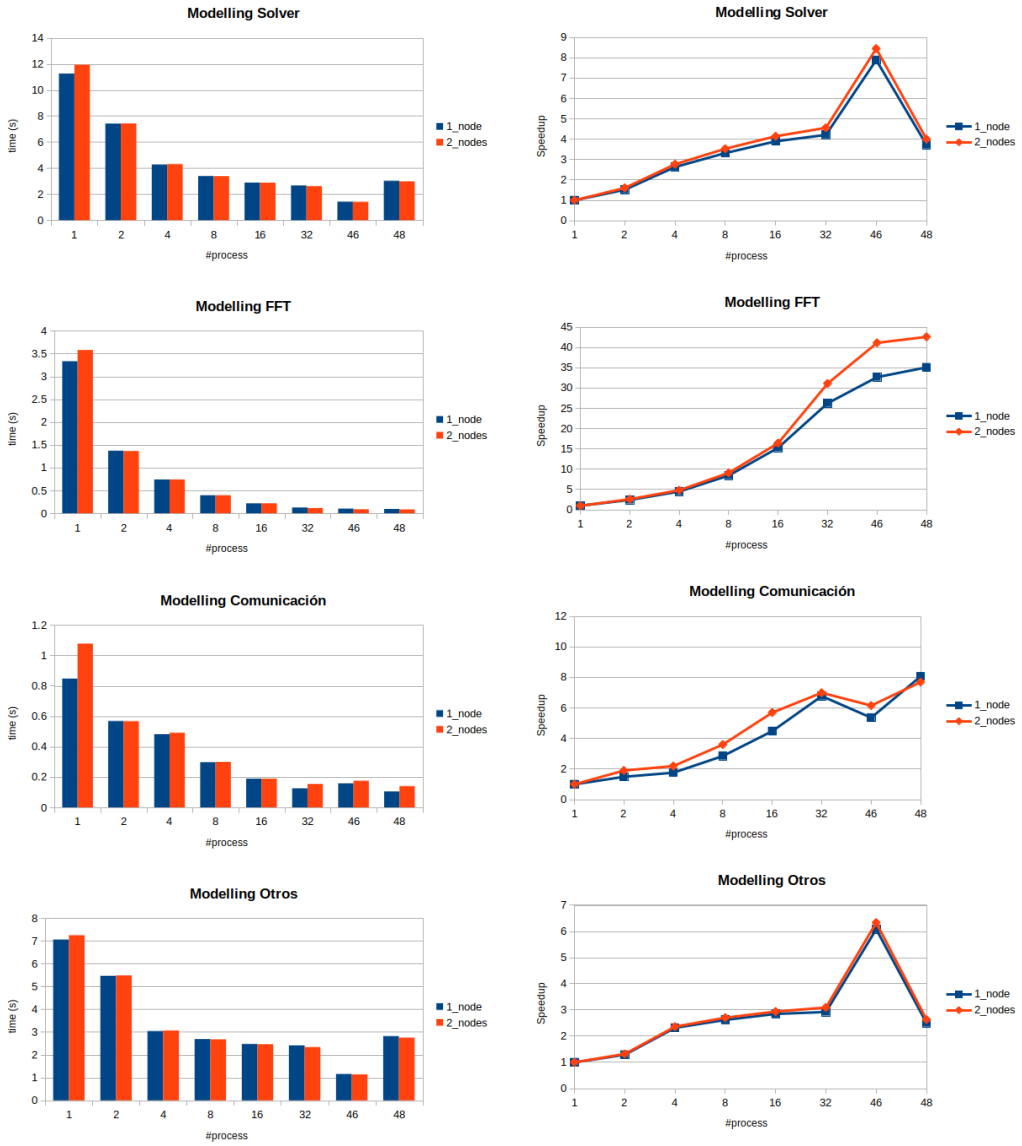


Figura 23: *Gráficas del tiempo de ejecución (izquierda) y speedup (derecha) del Solver 3D con el modo "modelling" con debug_snapshot_time_skip = 6. Elaboración propia.*