Universidad San Jorge

Escuela de Arquitectura y Tecnología

Grado en Ingeniería Informática

Proyecto Final

Computación voluntaria para la simulación de tejidos cardiacos heterogéneos

Autor del proyecto: Iván Royo Gutiérrez

Director del proyecto: Jesús Carro Fernández

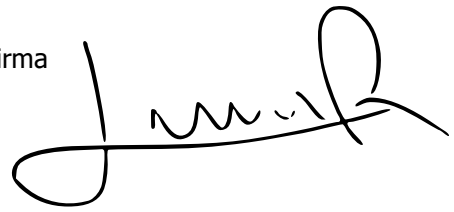
Villanueva de Gállego, 20 de junio de 2025



Este trabajo constituye parte de mi candidatura para la obtención del título de Graduado en Ingeniería Informática por la Universidad San Jorge y no ha sido entregado previamente (o simultáneamente) para la obtención de cualquier otro título.

Este documento es el resultado de mi propio trabajo, excepto donde de otra manera esté indicado y referido.

Doy mi consentimiento para que se archive este trabajo en la biblioteca universitaria de Universidad San Jorge, donde se puede facilitar su consulta.

Firma  Fecha: 20 de junio de 2025

Dedicatoria y Agradecimiento

A ti, papá, por ser siempre mi mano derecha para todo lo que hago, por escucharme siempre en todo lo que te cuento, y por haberme enseñado desde pequeño que todo buen esfuerzo tiene su merecida recompensa.

A ti, mamá, por hacerme ver siempre la vida de color rosa, por sacarme siempre una sonrisa aunque no sea mi mejor día, y por todos los valores tan buenos que me has inculcado a lo largo de esta vida.

A ti, Nerea, por ser mi compañera de vida, por aguantarme siempre que estoy enfadado, y por siempre estar ahí para cuando te necesito, a pesar de nuestras diferencias. Siempre voy a estar ahí para lo que necesites.

A ti, Paula, por haberme acompañado en todo este camino, por haber estado ahí siempre que te necesitaba, por haberme escuchado cuando nadie más quería hacerlo. Gracias a todo tu cariño, todo tu amor, y toda tu confianza que depositas en mí, y por haberme dado fuerzas cuando más lo he necesitado.

A mis tíos, Juan y Charo, por haber sido prácticamente como mis padres en esta época de mi vida, por haberme aguantado cuando no había sido el mejor de mis días, y por haberme tratado siempre como uno más.

A mis abuelos, ya que gracias a vosotros ese niño no hubiera sido la persona que es hoy en día, vosotros me habéis enseñado a ser la persona que soy. Y aunque algunos me habéis dejado por el camino, sé que allí donde estéis estaréis muy orgullosos de la persona que soy.

Al resto de mi familia cercana, gracias por vuestro apoyo y por los momentos vividos durante esta etapa de mi vida. Sin vosotros tampoco sería la persona que soy hoy.

A mis amigos más cercanos, por haber estado siempre ahí conmigo en lo malo y en lo bueno, por haberme ayudado a desconectar aun cuando creía que no podía. Gracias por todos esos buenos momentos que me habéis dado. Sin vosotros tampoco hubiera sido posible.

A ti, Jesús, director de este proyecto, por haberme enseñado este campo tan apasionante, y por haberme aguantado en todo momento. Han sido tres años maravillosos de recorrido a tu lado, y todo un honor para mí que hayas podido ser mi director de proyecto. Muchas gracias por tu apoyo, por tus consejos, y por aportarme esa gran sabiduría que tienes; para mí siempre serás un referente.

Tabla de contenido

Resumen *1*

Abstract *2*

1. Introducción *3*

1.1. Motivación **3**

1.2. Descripción **4**

2. Contexto, Antecedentes y Estado del Arte *7*

2.1. Electrofisiología Cardiaca **7**

2.1.1. Anatomía general del corazón y su relación con las arritmias 7

2.1.2. Sistema de conducción eléctrica del corazón 8

2.1.3. Potencial de acción cardiaco y su propagación 9

2.2. Evolución Histórica de la Modelización Computacional en Electrocardiología **10**

2.2.1. Modelo de Hodgkin y Huxley (1952): Bases de la electrofisiología computacional 10

2.2.2. Modelización del potencial de acción en fibras de Purkinje: El modelo de Noble (1962) . 11

2.2.3. Evolución histórica y modelos actuales en electrofisiología cardíaca 12

2.3. Computación Voluntaria en la Investigación Científica **13**

2.3.1. Origen y evolución de BOINC 13

2.3.2. Aplicaciones biomédicas de BOINC 15

2.4. Plataformas existentes relacionadas con la Simulación Cardiaca **16**

2.4.1. El simulador EMOS (Elvira): fundamentos, avances y proyección futura 16

2.4.2. CHASTE: un marco test-driven para la simulación fiable y escalable 16

2.4.3. openCARP: un entorno abierto para la simulación integral y multiescala en electrofisiología cardiaca 17

3. Objetivos *19*

4. Metodología *20*

4.1. Metodología SCRUM **20**

4.2. Metodología GitFlow **21**

4.3. Justificación de las Metodologías seleccionadas **21**

4.4. Adaptación de la Metodología de SCRUM para el Proyecto 22
4.4.1. Roles definidos en el proyecto 23
4.4.2. Eventos y estructura de los Sprints 23
4.5. Adaptación de la metodología GitFlow para el proyecto 23
4.6. Planificación Inicial del Proyecto 24
5. Análisis e Implementación 26
5.1. Análisis del problema de la propagación del impulso eléctrico en una fibra cardiaca 26
5.2. Simulador Denis Fiber 29
5.2.1. Archivo de configuración de la simulación 29
5.2.2. Simulación de una fibra cardiaca 32
5.2.3. Gestión del output 36
5.2.4. Postprocesado en los ordenadores voluntarios 37
5.2.5. Integración de la aplicación con BOINC 39
5.2.6. Dependencias de la aplicación 42
5.3. Integración en el proyecto de BOINC: DENIS@Home 44
5.3.1. Adaptaciones del servidor 44
5.3.2. Mandar tareas a los voluntarios 46
5.3.3. Validador de resultados 47
5.3.4. Asimilador de archivos 48
5.4. Entorno de pruebas con Docker Compose 49
5.4.1. Base de datos: MySQL 49
5.4.2. Ubuntu como servidor del proyecto 50
5.4.3. Script de inicialización del proyecto 52
5.5. Compilación multiplataforma y automatización 52
5.5.1. Imágenes de Docker para compilación en entornos Windows y Linux 52
5.5.2. Compilación para entornos macOS 53
5.5.3. Scripts de automatización de la compilación para las diferentes plataformas 54
6. Estudio Económico 56
6.1. Costes de la infraestructura 56
6.1.1. Costes del servidor actual del proyecto 56

6.1.2. Costes mensuales de un servidor en la nube 57
6.2. Costes de recursos humanos 58
6.3. Costes del material personal 59
6.3.1. Ordenador Personal 59
6.3.2. Licencia de IDE para el desarrollo 60
6.3.3. Resumen de costes 60
6.4. Costes totales del proyecto 61
7. Resultados y Evaluación 62
7.1. Validación frente a ELVIRA 63
7.2. Evaluación de checkpoints en simulaciones de larga duración 64
7.3. Test de variabilidad de la conductancia de la corriente Ks 66
7.4. Desviaciones a lo largo del proyecto 69
8. Conclusiones 72
8.1. Grado de cumplimiento de los objetivos planteados 72
8.2. Adaptaciones realizadas y aprendizajes durante el desarrollo 72
8.3. Posibles mejoras y futuras ampliaciones 73
8.3.1. Pruebas adicionales a medio/largo plazo 73
8.3.2. Mejoras con respecto al simulador 74
8.3.3. Mejoras con respecto al servidor de BOINC 75
8.4. Reflexión final 75
9. Bibliografía 76
10. Anexos 80

Tabla de figuras

Figura 1: Flujo de trabajo clínico basado en el concepto de 'digital twin' en cardiología. Obtenida de [5] 4
Figura 2: Componentes e interfaces RPC del sistema BOINC. Obtenida de [7] 5
Figura 3: Representación del sistema de conducción cardiaco. Obtenida de [6] 8
Figura 4: Corrientes y Canales implicados en la generación del Potencial de Reposo y de Acción. Obtenido de [12] 10
Figura 5: Circuito eléctrico que representa la membrana del axón de calamar. Obtenido de [13] 11
Figura 6: Circuito eléctrico equivalente para la membrana de la fibra de Purkinje. Obtenido de [14] 11
Figura 7: Diagrama esquemático de la dinámica del modelo ventricular de Luo-Rudy. Obtenido de [11] 12
Figura 8: Diagrama de Gantt sobre la Planificación Inicial 25
Figura 9: Tablero del proyecto para la aplicación de SCRUM 25
Figura 10: Vista general de la implementación del sistema completo. 26
Figura 11: Ejemplo archivo básico de configuración XML 32
Figura 12: Método encargado de abordar la simulación de la fibra cardiaca 34
Figura 13: Fragmento encargado de calcular la propagación del voltaje y actualizar los estados 35
Figura 14: Ejemplo de estructura de un archivo de salida (output) 37
Figura 15: Comando usado para mostrar la estructura de carpetas requerida por el simulador 43
Figura 16: Daemons correspondientes al validador y asimilador respectivamente 45
Figura 17: Ejemplo de la estructura de carpetas que debe de seguir una aplicación de BOINC 45
Figura 18: Comandos para registrar y crear un nuevo trabajo en el servidor de BOINC 46
Figura 19: Carpeta sample_results con los resultados correspondientes de la prueba de mandar tareas 47
Figura 20: Comparación del potencial de acción en la posición 50 a partir de los resultados obtenidos en ELVIRA y DENIS 63
Figura 21: Resumen de estados y resultados del lote de 10.000 simulaciones (panel de administración). 64
Figura 22: Task Outcome Distribution – Éxitos (verde) vs. errores de cómputo (rojo). 65
Figura 23: Validation State Distribution – Estado inicial (gris), validado (verde) e inválido (rojo). 65
Figura 24: Client Error Distribution – Errores de cómputo y tareas abortadas por el usuario. 66

Figura 25: Histogramas de la distribución de APD10, APD25, APD50, APD75 y APD90 para la posición 50 de ambos experimentos (primera imagen prueba de 10s, segunda de 50s). 67
Figura 26: Medias y desviaciones típicas de los histogramas (respectivamente) del APD para la posición 50.....68
Figura 27: Potenciales de acción obtenidos de la prueba, superpuestos en la posición 50 frente a la simulación base. Primera correspondiente con la prueba de 10s, segunda con la de 50s. 68
Figura 28: Gráficas de progreso de la prueba a lo largo del tiempo desde su lanzamiento hasta su fin. La primera correspondiente a las simulaciones de 10s y la segunda a las de 50s. 69
Figura 29: Componentes necesarios para la compilación BOINC con VS Build Tools. 96

Tabla de tablas

Tabla 1: Desglose de costes del servidor que alberga BOINC.....56
Tabla 2: Desglose de costes mensuales del servidor en AWS.....58
Tabla 3: Desglose de costes de recursos humanos (SS incluida).....59
Tabla 4: Resumen de costes de material personal.....60
Tabla 5: Desglose de costes totales para el proyecto.....61

Resumen

El objetivo principal de este Trabajo Fin de Grado es ampliar la plataforma DENIS para que sea capaz de ejecutar simulaciones multiescala de electrofisiología cardíaca. En la actualidad, DENIS permite realizar hasta 60.000 simulaciones diarias de células cardíacas aisladas, pero no abarca modelos más complejos de tejido o fibras que reflejen mejor la fisiología real del corazón. Esta mejora busca dar respuesta a la creciente demanda de estudios que exploren la propagación del impulso eléctrico en tejidos de distinta morfología o composición iónica, tanto en condiciones normales como en entornos patológicos.

En este proyecto se ha desarrollado un nuevo simulador capaz de manejar modelos 1D con características celulares heterogéneas a lo largo de la fibra y de calcular indicadores clave (por ejemplo, APD90 o APD50) directamente en los ordenadores voluntarios.

Gracias a estas mejoras, se prevé que DENIS pueda impulsar la investigación en electrofisiología computacional, permitiendo profundizar en el estudio de la actividad eléctrica del corazón y los fenómenos fisiopatológicos asociados, al tiempo que se optimizan los recursos distribuidos ofrecidos por miles de voluntarios.

Palabras clave: computación voluntaria, electrofisiología cardíaca, DENIS, arritmias, simulación multiescala.

Abstract

The main objective of this Final Degree Project is to extend the DENIS platform to support multiscale cardiac electrophysiology simulations. Currently, DENIS can run up to 60,000 daily simulations of isolated cardiac cells but does not cover more complex tissue or fiber models that more closely reflect real heart physiology. This enhancement aims to meet the growing need for studies exploring electrical impulse propagation in tissues of varying morphology or ionic composition, under both normal and pathological conditions.

The proposed solution includes a new simulator capable of handling 1D models with heterogeneous characteristics in the cells and calculating key markers (e.g., APD90 or APD50) directly on volunteer computers.

With these improvements, DENIS is expected to significantly boost research in computational electrophysiology, enabling a deeper examination of the heart's electrical activity and related pathophysiological phenomena, while optimizing the distributed resources provided by thousands of volunteers.

Keywords: volunteer computing, cardiac electrophysiology, DENIS, arrhythmias, multiscale simulation

1. Introducción

1.1. Motivación

Las enfermedades cardiovasculares son la principal causa de mortalidad mundial. Según la Organización Mundial de la Salud, este tipo de enfermedades son la causa de fallecimiento de aproximadamente 17,9 millones de personas al año, lo que correspondería a un 32% de la mortalidad global [1]. Dentro de las enfermedades cardiovasculares, la Insuficiencia Cardiaca (IC) es una de las más frecuentes entre las personas que sufren este tipo de enfermedades. Su incidencia está creciendo debido al inevitable envejecimiento de la población, y al aumento de la esperanza de vida media, haciendo que esta sea una de las principales causas de hospitalización en personas mayores de 65 años, conllevando un deterioro progresivo de la calidad de vida, y, según la Sociedad Española de Cardiología, entre un 2% y 3% de los gastos sanitarios españoles. La Insuficiencia Cardiaca (IC) se define como la incapacidad del corazón para suministrar oxígeno a una frecuencia acorde con las necesidades. Aproximadamente, entre el 1-2% de la población adulta de los países desarrollados padece IC, con una prevalencia que se incrementa hasta más del 10% en personas mayores de 70 años. Además, la mortalidad asociada sigue siendo alarmante: cerca del 50% de los pacientes fallecen en los cinco años posteriores al diagnóstico, debido al fallo progresivo de bomba o a arritmias ventriculares que pueden derivar en muerte súbita [2].

La simulación computacional de la actividad eléctrica de las células cardíacas se ha convertido en una herramienta fundamental para la comprensión de diversos procesos fisiológicos y patológicos, en particular en el estudio de las arritmias cardíacas. Desde las primeras aplicaciones de las ecuaciones de Hodgkin y Huxley a la electrofisiología cardíaca, los modelos han evolucionado para incluir una representación detallada de los canales iónicos, las interacciones celulares y los mecanismos subyacentes a las alteraciones del ritmo cardíaco. Estos avances han permitido simular con precisión la dinámica eléctrica del corazón, facilitando la identificación de factores desencadenantes de arritmias y la evaluación de estrategias terapéuticas, consolidando así la modelización computacional como una herramienta indispensable en la investigación cardiovascular [3].

Actualmente, existen modelos computacionales para casi todos los tipos de células del corazón, incluidas las nodales, auriculares, ventriculares y de Purkinje, permitiendo simular con precisión su actividad eléctrica y funcional. Estos modelos han evolucionado para reflejar mejor la fisiología cardíaca en diferentes condiciones, ayudando a comprender la heterogeneidad de la actividad eléctrica y su relación con las arritmias [4]. Son muchos los estudios que han utilizado este tipo de herramientas para aclarar los mecanismos de generación de arritmias cardiacas, como por

ejemplo el estudio de Corral-Acero et al. (2020), que habla sobre el concepto de "digital twin" en cardiología. Este enfoque combina modelos computacionales mecanicistas y estadísticos para integrar datos clínicos en tiempo real, facilitando el análisis personalizado del comportamiento cardíaco. Su aplicación permite predecir la respuesta a tratamientos y mejorar la precisión del diagnóstico en el ámbito de la cardiología de precisión, avanzando en la comprensión de las arritmias y su manejo clínico [5]. La evolución de estos modelos ha sido clave en la simulación de la heterogeneidad eléctrica del miocardio, permitiendo analizar la propagación de impulsos eléctricos en diferentes regiones del corazón y su impacto en la generación de arritmias [6]. Además, su aplicación en la investigación biomédica ha facilitado la evaluación de fármacos, el estudio de la variabilidad interindividual en la respuesta terapéutica y el diseño de estrategias personalizadas para el tratamiento de enfermedades cardiovasculares [6].

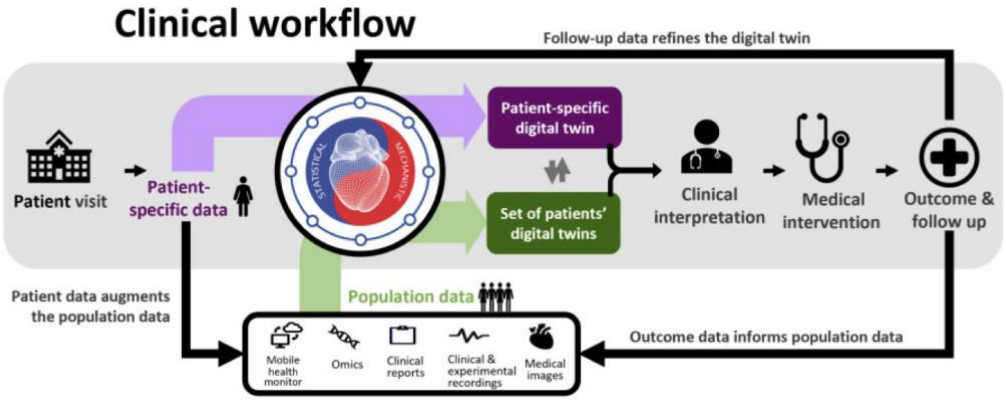


Figura 1: Flujo de trabajo clínico basado en el concepto de 'digital twin' en cardiología. Obtenida de [5]

Si bien existen modelos computacionales para cada tipo de célula cardíaca, la validación de estos modelos sigue siendo un desafío clave. Los modelos celulares se ajustan utilizando marcadores electrofisiológicos pero estos valores pueden diferir cuando se comparan células aisladas con tejidos debido a la interacción entre células adyacentes [6]. El músculo cardíaco se distingue por su capacidad de propagación del impulso eléctrico en múltiples direcciones, lo que introduce complejidades que no siempre se reflejan en simulaciones de célula aislada [6]. Estas diferencias subrayan la importancia de validar los modelos computacionales con datos obtenidos de tejidos cardíacos para mejorar su precisión y aplicabilidad en la investigación biomédica.

1.2. Descripción

DENIS (Distributed Computing, Electrophysiological Models, Networking Collaboration, In Silico Research, Sharing Knowledge) es una iniciativa que surgió en la Universidad San Jorge y que, desde su lanzamiento en 2015, ha permitido realizar simulaciones de electrofisiología cardíaca a través de una red distribuida de ordenadores voluntarios [8] [9].

Por un lado, DENIS utiliza el estándar CellML en su versión 1.1 para describir los modelos de electrofisiología cardíaca empleados en las simulaciones. CellML es un estándar abierto basado en XML, desarrollado por el Auckland Bioengineering Institute de la Universidad de Auckland junto con grupos de investigación afiliados. Su uso facilita enormemente la inclusión y la interoperabilidad de distintos modelos matemáticos de células cardíacas entre investigadores. Este estándar permite definir claramente la estructura matemática del modelo, las variables implicadas y sus relaciones, ofreciendo un marco unificado que simplifica la integración y reutilización de modelos en plataformas de simulación como DENIS [9] [10].

Por otro lado, el proyecto DENIS se apoya en la plataforma BOINC (Berkeley Open Infrastructure for Network Computing), un middleware de código abierto desarrollado en la Universidad de California, Berkeley, que permite a investigadores de diversas áreas acceder a grandes cantidades de recursos computacionales sin necesidad de infraestructuras dedicadas de alto coste [7].

BOINC funciona mediante un sistema distribuido en el que los cálculos son ejecutados en dispositivos de usuarios voluntarios, coordinados a través de un servidor central de proyectos. El BOINC project server gestiona las simulaciones enviando fragmentos de trabajo a los ordenadores conectados, los cuales ejecutan los cálculos mediante un software llamado BOINC client. Una vez finalizadas las tareas, los resultados son enviados de vuelta al servidor para su recopilación y análisis. Además, BOINC permite la gestión de cuentas y la asignación de créditos a los voluntarios según su contribución computacional, facilitando así una infraestructura escalable para proyectos científicos de gran envergadura [7].

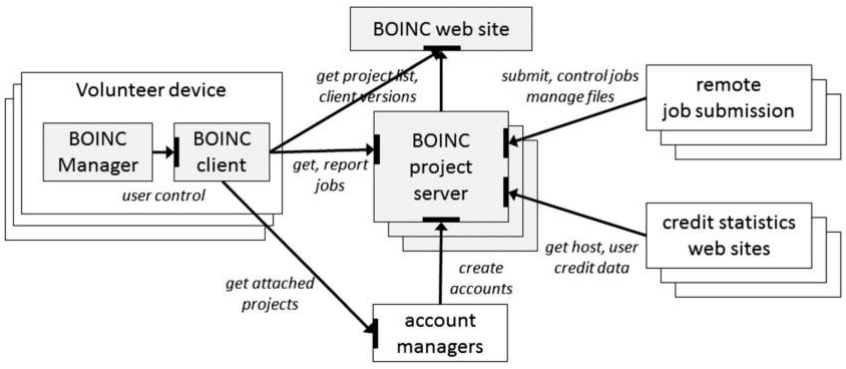


Figura 2: Componentes e interfaces RPC del sistema BOINC. Obtenida de [7]

En el contexto de DENIS, esta plataforma ha permitido la realización de simulaciones computacionales de la actividad eléctrica cardíaca, alcanzando una capacidad de hasta 60.000 simulaciones diarias de una hora de duración [8]. A pesar del éxito de DENIS en la simulación de células cardíacas aisladas, un desafío clave en el ámbito de la electrofisiología computacional es

la expansión hacia modelos multiescala, donde no solo se simulen células individuales, sino también tejidos cardíacos más complejos. Esto permitiría estudiar la propagación de señales eléctricas en fibras heterogéneas, proporcionando una representación más realista de la actividad cardíaca en condiciones fisiológicas y patológicas. Sin embargo, la implementación de simulaciones multiescala requiere importantes adaptaciones en la infraestructura de DENIS, optimizando la gestión de tareas distribuidas y aumentando la capacidad de procesamiento de la plataforma.

Dado que la simulación de sistemas multicelulares supone un elevado coste computacional, el uso de BOINC se presenta como una alternativa eficiente para distribuir la carga de trabajo entre miles de dispositivos conectados a la red. Gracias a su diseño modular y escalable, BOINC puede gestionar el procesamiento de simulaciones de gran escala sin necesidad de depender de centros de computación de alto rendimiento (HPC), permitiendo la realización de estudios que de otro modo serían inalcanzables debido a los costes de infraestructura [7].

En este contexto, el presente trabajo aborda el desafío de extender la plataforma DENIS para facilitar la ejecución de simulaciones en estructuras cardíacas más complejas, superando la actual limitación de simulaciones de células aisladas. La expansión del sistema permitirá mejorar la capacidad de análisis en electrofisiología computacional, proporcionando una herramienta más potente y versátil para la investigación en el campo de la modelización cardíaca. Con ello, DENIS podrá consolidarse como una plataforma clave para la simulación de la actividad eléctrica del corazón, facilitando el estudio de patologías como las arritmias y la insuficiencia cardíaca en entornos virtuales [8] [9].

2. Contexto, Antecedentes y Estado del Arte

2.1. Electrofisiología Cardiaca

2.1.1. Anatomía general del corazón y su relación con las arritmias

Para comprender la actividad eléctrica del corazón y su relación con las arritmias, resulta fundamental conocer la estructura anatómica y funcional del tejido muscular cardíaco (miocardio). Los modelos computacionales utilizados en electrofisiología cardíaca no solo dependen de ecuaciones matemáticas que describen la dinámica iónica de las células, sino que también requieren información detallada sobre la morfología y organización del tejido. La propagación del impulso eléctrico en el corazón está condicionada por la disposición de sus cavidades, la presencia de tabiques y estructuras valvulares, y la distribución de distintos tipos celulares, como células nodales, auriculares, ventriculares y de Purkinje [4]. Estas características anatómicas determinan los patrones de conducción y las posibles alteraciones que pueden desembocar en arritmias. Por ello, antes de profundizar en el análisis de los modelos computacionales, resulta imprescindible revisar la anatomía del corazón para contextualizar adecuadamente los procesos eléctricos que regulan su funcionamiento.

El corazón es un órgano muscular situado en la cavidad torácica, dividido en cuatro cavidades (dos aurículas y dos ventrículos) por septos y válvulas que regulan el flujo sanguíneo. A nivel histológico, su pared se compone de tres capas principales: endocardio (capa interna), miocardio (capa media responsable de la contracción) y epicardio (capa más externa). Además, el corazón posee un sistema especializado de conducción (nodo SA, nodo AV, haz de His y fibras de Purkinje) que coordina la activación secuencial de aurículas y ventrículos [11].

Desde un punto de vista bioeléctrico, la disposición de las fibras musculares y la presencia de tabiques y tejidos de conducción dotan al miocardio de propiedades anisotrópicas, es decir, la velocidad de conducción puede variar según la orientación de las fibras. Esta organización anatómica, junto con la expresión diferencial de uniones comunicantes y canales iónicos, facilita que el impulso eléctrico se propague de modo relativamente uniforme en condiciones normales (aunque con velocidades distintas en dirección longitudinal y transversal), pero puede también dar lugar a desacoplamientos locales o rutas de conducción lenta si existen discontinuidades estructurales o funcionamientos iónicos anómalos [11].

Las arritmias pueden emerger cuando esta propagación sufre bloqueos unidireccionales o recorridos en los que la onda de excitación "reentra" de forma repetitiva en una región ya parcialmente recuperada, cerrando un circuito de activación. El trabajo de Kleber y Rudy [11] destaca que la presencia de obstáculos anatómicos (por ejemplo, tejido fibroso) o la

heterogeneidad en la expresión de canales iónicos y uniones intercelulares pueden fomentar dichos bloqueos y reentradas, dando lugar a taquicardias y fibrilaciones. Asimismo, la geometría del miocardio (por ejemplo, trabéculas o cambios bruscos de fibra) y la forma de las cavidades cardiacas influyen en fenómenos como la colisión de ondas de despolarización, la aparición de zonas de enlentecimiento y la formación de espirales rotatorias con posible perpetuación de la actividad arrítmica.

2.1.2. Sistema de conducción eléctrica del corazón

Como se ha mencionado anteriormente, el corazón dispone de un sistema de conducción especializado que permite la activación ordenada de sus cavidades. El funcionamiento del corazón está íntimamente ligado a su actividad bioeléctrica. Su contracción rítmica y sincronizada es posible gracias a la propagación de impulsos eléctricos a lo largo del sistema de conducción cardíaco, que coordina la activación del miocardio para garantizar un bombeo eficiente de la sangre [6].

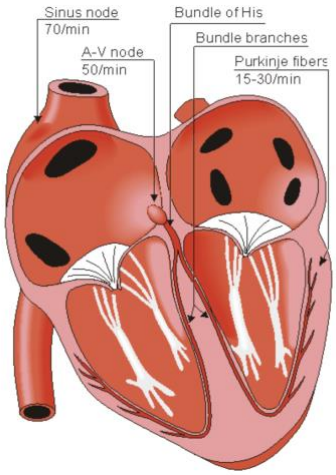


Figura 3: Representación del sistema de conducción cardiaco. Obtenida de [6]

El **nodo sinoauricular (SA)**, ubicado en la aurícula derecha, genera el impulso eléctrico primario, estableciendo el ritmo cardíaco mediante la activación espontánea de sus células. Este impulso se propaga a través del miocardio auricular hasta alcanzar el **nodo auriculoventricular (AV)**, donde se experimenta un retraso que permite que las aurículas completen su contracción antes de que los ventrículos se activen. Desde el nodo AV, la señal viaja por el Haz de His, que se divide en las ramas derecha e izquierda, extendiéndose por las fibras de Purkinje hasta alcanzar los ventrículos, permitiendo su contracción simultánea [6].

2.1.3. Potencial de acción cardiaco y su propagación

Desde el punto de vista electrofisiológico, la activación del miocardio se basa en la generación de señales eléctricas que regulan la contracción del corazón. Este proceso es posible gracias a la generación y propagación del potencial de acción, que permite la activación secuencial del tejido cardíaco.

El potencial de acción cardíaco sigue una secuencia bien definida de fases, dependiendo del tipo de célula cardíaca en la que se genere. En las células del miocardio ventricular, este fenómeno se divide en cinco fases principales [12]:

- **Fase 0 - Upstroke o despolarización rápida**: Se inicia cuando un estímulo despolarizante o un potencial generado espontáneamente alcanza el umbral necesario para desencadenar un potencial de acción. En las células auriculares, ventriculares y del sistema His-Purkinje, esta fase es impulsada por la apertura de canales de sodio (Na^+) dependientes de voltaje, permitiendo una entrada rápida de Na^+ y generando un aumento abrupto del potencial de membrana. En las células del nodo SA y nodo AV, la despolarización es impulsada por la corriente de calcio (Ca^{2+}) a través de canales de calcio activados por voltaje [12].
- **Fase 1 - Repolarización inicial rápida**: Tras la despolarización, la membrana celular experimenta una repolarización transitoria debido a la inactivación de los canales de Na^+ y la activación de una corriente transitoria de salida de potasio (K^+), conocida como I_{to} (transient outward current). Esta salida de carga positiva genera una leve disminución del potencial de membrana, preparando a la célula para la fase de meseta [12].
- **Fase 2 - Meseta**: Se produce un equilibrio entre la entrada de calcio (Ca^{2+}) a través de canales de calcio tipo L ($I_{Ca,L}$) y la salida de potasio (K^+), manteniendo la membrana en un voltaje elevado por un período prolongado. Esta fase es crucial para la contracción del miocardio, ya que el calcio liberado en este período activa el acoplamiento excitación-contracción. La duración de esta fase varía según el tipo celular, siendo más prolongada en las fibras ventriculares que en las auriculares [12].
- **Fase 3 - Repolarización rápida final**: Se caracteriza por la desactivación progresiva de los canales de calcio y la activación de corrientes de potasio retardadas (I_{Kr} e I_{Ks}). La salida neta de cargas positivas debido a estos canales devuelve el potencial de membrana a valores cercanos al estado de reposo. Esta repolarización restablece las condiciones iónicas iniciales y prepara a la célula para un nuevo ciclo de activación [12].
- **Fase 4 - Potencial de membrana en reposo y despolarización diastólica**: Durante esta fase, la célula mantiene su potencial de reposo en valores de -50 a -95 mV, dependiendo del tipo celular. La estabilidad de esta fase depende principalmente de la

permeabilidad al potasio (K^+), regulada por los canales I_{K1} (corriente rectificadora de entrada). En células marcapasos, como las del nodo SA, la despolarización diastólica permite alcanzar el umbral y generar un nuevo potencial de acción de manera espontánea, facilitando la automatización del ritmo cardíaco [12].

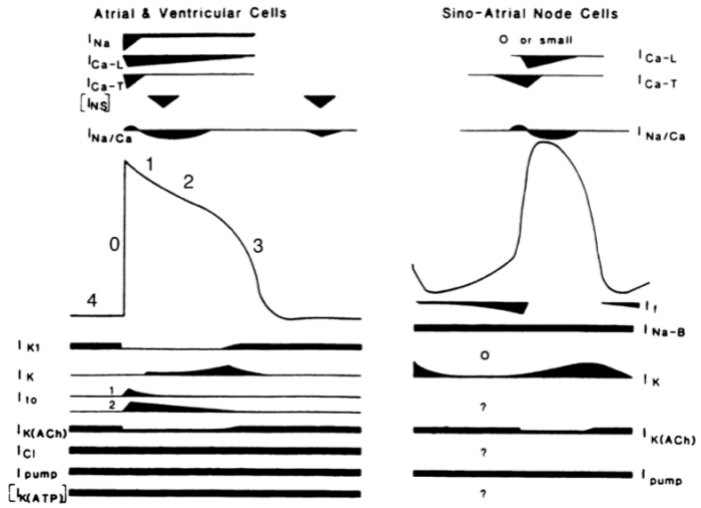


Figura 4: Corrientes y Canales implicados en la generación del Potencial de Reposo y de Acción. Obtenido de [12]

2.2. Evolución Histórica de la Modelización Computacional en Electrocardiología

2.2.1. Modelo de Hodgkin y Huxley (1952): Bases de la electrofisiología computacional

En 1952, Alan L. Hodgkin y Andrew F. Huxley publicaron un estudio en el que describieron de manera cuantitativa los mecanismos bioeléctricos responsables de la generación y propagación del potencial de acción en el axón gigante de calamar (Loligo). Este trabajo, basado en experimentos con la técnica de voltage-clamp, permitió caracterizar las corrientes iónicas que atraviesan la membrana celular durante la excitación neuronal, estableciendo los principios fundamentales de la modelización computacional en electrofisiología [13].

El modelo propuesto por Hodgkin y Huxley estableció que la propagación del impulso nervioso se debe a cambios en la conductancia de dos iones principales, sodio (Na^+) y potasio (K^+), los cuales atraviesan la membrana mediante canales iónicos específicos cuya apertura y cierre dependen del voltaje. A través de un sistema de ecuaciones diferenciales no lineales, describieron con precisión la variación de estas conductancias a lo largo del tiempo y su influencia en la evolución del potencial de membrana [13].

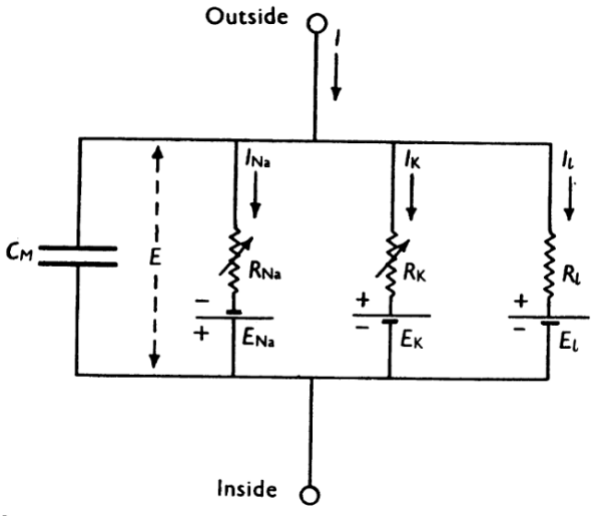


Figura 5: Circuito eléctrico que representa la membrana del axón de calamar. Obtenido de [13]

2.2.2. Modelización del potencial de acción en fibras de Purkinje: El modelo de Noble (1962)

En 1962, Denis Noble publicó un artículo en el cual adaptó las ecuaciones originales de Hodgkin y Huxley para describir con precisión la actividad eléctrica de las fibras de Purkinje cardíacas. Este trabajo representó un avance significativo en la comprensión de la electrofisiología cardíaca, proporcionando un modelo matemático que captura las características distintivas de estas células especializadas.

Noble modificó las ecuaciones de Hodgkin y Huxley para reflejar las propiedades únicas de las fibras de Purkinje, incluyendo su capacidad de generar potenciales marcapasos y su respuesta a diferentes estímulos eléctricos [14].

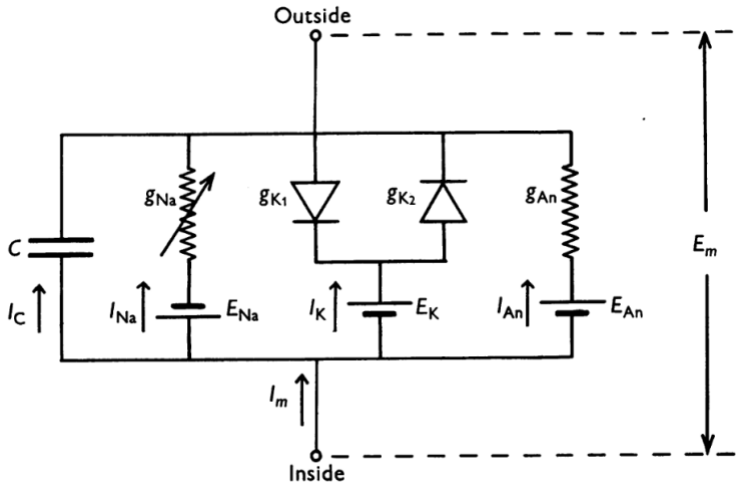


Figura 6: Circuito eléctrico equivalente para la membrana de la fibra de Purkinje. Obtenido de [14]

2.2.3. Evolución histórica y modelos actuales en electrofisiología cardíaca

La modelización computacional de la electrofisiología cardiaca ha recorrido un largo camino desde los trabajos pioneros de Hodgkin y Huxley (1952), cuyo modelo describía de forma cuantitativa la generación y propagación del potencial de acción fundamentalmente a partir de tres corrientes iónicas (Na^+, K^+ y una corriente de fuga). Posteriormente, Noble (1962) amplió estas ecuaciones para adaptarlas al comportamiento de las fibras de Purkinje, incorporando aspectos como la automatización y el papel de canales específicos en el tejido cardíaco.

A lo largo de las décadas siguientes, se fueron añadiendo progresivamente más corrientes y complejidad en la descripción de los canales iónicos, pasando de unas pocas ecuaciones diferenciales, a modelos capaces de incluir más de 15-20 corrientes diferentes. Por ejemplo, el modelo de Beeler y Reuter (1977) [15] introdujo cuatro corrientes principales y prestó especial atención al rol del calcio (Ca^{2+}) en la fase de meseta, mientras que Luo y Rudy (1991, 1994) [15] [16] refinaron la descripción de la corriente rápida de sodio (I_{Na}) e integraron procesos como el intercambio Na^+-Ca^{2+} y la bomba sodio-potasio. Estas mejoras permitieron simular con mayor precisión la duración del potencial de acción, la repolarización y la homeostasis intracelular de iones.

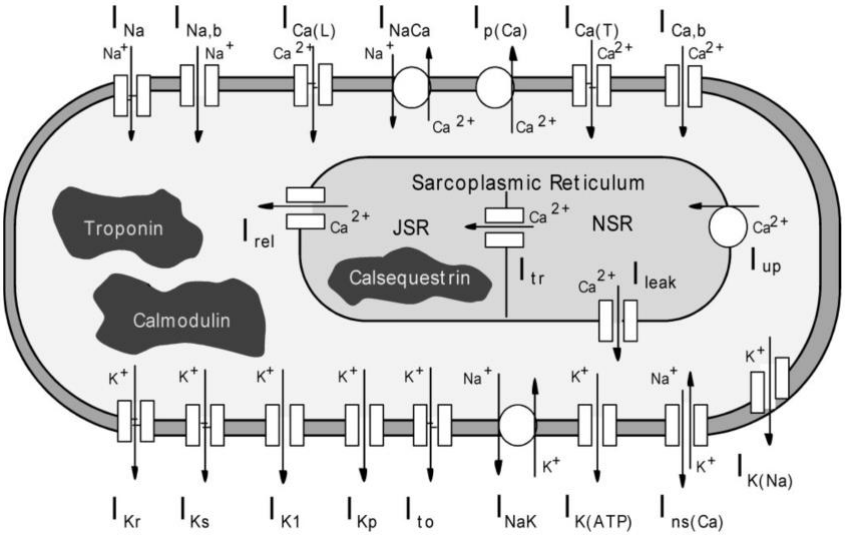


Figura 7: Diagrama esquemático de la dinámica del modelo ventricular de Luo-Rudy. Obtenido de [11]

Posteriormente aparecieron planteamientos más completos capaces de reproducir mejor los fenómenos de reentrada y la dinámica del calcio intracelular. El trabajo de ten Tusscher y Panfilov (2006) [18] abordó la aparición de alternancia (*alternans*) y la fragmentación de espirales que conduce a fibrilación ventricular, haciendo hincapié en la restitución de la duración del potencial de acción y el efecto de la recuperación de la corriente de sodio sobre la estabilidad eléctrica del

tejido. Posteriormente, Grandi et al. (2010) [19] presentaron un nuevo enfoque para la modelización de la célula ventricular humana que incorporó una descripción detallada de la corriente lenta de calcio y de la cinética intracelular del calcio, integrando mejor la relación entre el potencial de acción y la contracción. Por su parte, O'Hara et al. (2011) [20] propusieron un modelo orientado específicamente al comportamiento del miocardio humano sano, validándolo con múltiples protocolos y reproduciendo fenómenos como la dispersión transmural de la repolarización o la aparición de posdespolarizaciones tempranas. Finalmente, Tomek et al. (2019) [21] introdujeron una reformulación de estos planteamientos incorporando modificaciones en la fuerza motriz del calcio, la descripción de la corriente rápida de potasio y el balance intracelular de sodio, logrando un ajuste más cercano a las respuestas experimentales frente a bloqueos iónicos y condiciones patológicas, sin dejar de reflejar la complejidad del acoplamiento excitación-contracción en el tejido ventricular humano.

2.3. Computación Voluntaria en la Investigación Científica

2.3.1. Origen y evolución de BOINC

La computación voluntaria ha sido un elemento clave en la evolución de la computación distribuida, permitiendo que miles de usuarios contribuyan con la capacidad de sus ordenadores personales a la resolución de problemas científicos complejos. Su aplicación ha sido particularmente relevante en proyectos donde la cantidad de datos a procesar es inmensa y los recursos computacionales tradicionales resultarían insuficientes o demasiado costosos [22].

Uno de los primeros y más emblemáticos proyectos en este ámbito fue SETI@Home [24], lanzado en 1999 por un equipo de investigadores de la University of California, Berkeley, liderado por David P. Anderson. El proyecto nació como una iniciativa dentro del programa Search for Extraterrestrial Intelligence (SETI), cuyo objetivo era analizar señales de radio provenientes del espacio en busca de patrones que pudieran indicar la presencia de inteligencia extraterrestre. Para ello, se utilizaron datos obtenidos por el radiotelescopio de Arecibo, una de las instalaciones más grandes del mundo en su época, los cuales debían ser procesados para descartar interferencias y aislar posibles señales de origen no natural [24].

El desafío principal de SETI@Home radicaba en la enorme cantidad de datos recopilados: los radiotelescopios generan volúmenes masivos de información en tiempo real, los cuales, de procesarse en centros de supercomputación convencionales, requerirían costos extremadamente elevados y tiempos de cálculo poco viables para la comunidad científica. Ante esta problemática, los investigadores propusieron un modelo innovador: dividir el trabajo en pequeñas unidades de datos y distribuirlo entre miles de ordenadores personales conectados a internet. Cada voluntario instalaba un software cliente que descargaba un fragmento de datos, lo analizaba en segundo

plano utilizando el tiempo de inactividad del procesador, y posteriormente enviaba los resultados al servidor central para su integración en el análisis global [24].

La adopción de este sistema marcó un hito en la computación distribuida, convirtiendo a SETI@Home en uno de los proyectos científicos más populares de su época. Millones de usuarios en todo el mundo participaron, aportando una capacidad computacional agregada que rivalizaba con la de las supercomputadoras más avanzadas de finales de los años 90. Sin embargo, conforme el proyecto crecía, se evidenciaron ciertas limitaciones en su diseño. Entre los principales desafíos se encontraban la gestión eficiente de recursos, la escalabilidad del sistema y la posibilidad de aplicar este modelo a otros proyectos científicos más allá de SETI [24].

Para abordar estos desafíos, en 2002, David P. Anderson y su equipo en Berkeley desarrollaron BOINC (Berkeley Open Infrastructure for Network Computing), una plataforma de código abierto diseñada para gestionar múltiples proyectos de computación voluntaria dentro de una misma infraestructura [23].

BOINC no solo optimizó la gestión de tareas y la asignación de recursos, sino que también permitió la integración de múltiples proyectos científicos dentro de una infraestructura común, haciendo que cualquier investigador con necesidades computacionales pudiera acceder a una red global de voluntarios. Esto supuso un cambio fundamental en la computación distribuida, ya que se eliminó la necesidad de desarrollar un software específico para cada proyecto, ofreciendo en su lugar una plataforma estandarizada y flexible [23].

Desde un punto de vista técnico, BOINC introdujo importantes avances en seguridad, validación de datos y escalabilidad. Uno de los problemas de SETI@Home era la posibilidad de recibir resultados erróneos o manipulados por usuarios malintencionados. Para solucionar esto, BOINC implementó un sistema de validación redundante, en el cual múltiples dispositivos procesaban la misma unidad de trabajo y se comparaban los resultados antes de ser aceptados. Además, el sistema permitía la asignación dinámica de tareas, priorizando aquellas que requerían mayor urgencia y optimizando el uso del ancho de banda para minimizar la carga en los servidores centrales [23].

Otra innovación significativa fue la capacidad de BOINC para adaptarse a distintos tipos de hardware y sistemas operativos. Mientras que SETI@Home estaba diseñado principalmente para arquitecturas x86 en entornos Windows, BOINC permitió la ejecución de proyectos en Linux, MacOS y arquitecturas ARM, facilitando su adopción en una variedad de dispositivos, incluyendo servidores, ordenadores personales y, más recientemente, dispositivos móviles y microprocesadores embebidos [22].

Desde su lanzamiento, BOINC ha sido adoptado por numerosos proyectos científicos en áreas tan diversas como la astrofísica, la climatología, la biomedicina y la química computacional. Su capacidad para aprovechar la potencia computacional distribuida ha permitido la ejecución de simulaciones complejas a una escala que antes solo era posible en supercomputadores de alto rendimiento. A lo largo de los años, la plataforma ha continuado evolucionando, integrando nuevas tecnologías como el uso de unidades de procesamiento gráfico (GPU) y la compatibilidad con infraestructuras de computación en la nube, ampliando así su potencial de aplicación [22].

2.3.2. Aplicaciones biomédicas de BOINC

BOINC ha servido de base para un gran número de iniciativas científicas en el ámbito biomédico. Un listado completo de proyectos de computación voluntaria, incluidas sus áreas de investigación, se encuentra disponible en la web oficial de BOINC (https://boinc.berkeley.edu/projects.php). A pesar de que hay muchos proyectos relacionados con el ámbito biomédico, a continuación, se detallaran algunos de los más relevantes dentro de este.

Rosetta@Home: predecir estructuras de proteínas.

Rosetta@home se concibe como un proyecto de computación voluntaria que emplea la infraestructura BOINC para abordar el desafío de predecir la estructura de proteínas a partir de sus secuencias primarias. Desde sus inicios, el principal objetivo del proyecto ha sido combinar una función de energía físico-química de alta resolución con el poder computacional distribuido de miles de ordenadores personales, de modo que las proteínas puedan ser modeladas y refinadas a escala atómica. Esta iniciativa surge de la necesidad de llevar a cabo un muestreo intensivo de conformaciones, algo inalcanzable con los recursos limitados de muchos laboratorios, pero factible si se aprovecha la colaboración masiva de voluntarios. La participación en Rosetta@home no solo permite realizar predicciones más precisas de estructuras proteicas, sino que también promueve la divulgación científica y el interés de la sociedad en la biología estructural [25].

GPUGrid como plataforma de computación voluntaria para la biomedicina

GPUGrid surge como una iniciativa para aprovechar la extraordinaria capacidad de cálculo de las tarjetas gráficas (GPUs) en la ejecución de simulaciones intensivas de dinámica molecular a gran escala. Esta plataforma se apoya en BOINC con el fin de que miles de usuarios voluntarios donen el tiempo de inactividad de sus GPU para ejecutar cálculos científicos de forma masiva [26]. Al centrarse en tarjetas gráficas generalistas y programables, GPUGrid puede trabajar con un volumen de operaciones notablemente superior al de muchas soluciones tradicionales de CPU. Esto reduce de forma significativa los tiempos de simulación, facilitando la exploración de procesos biomoleculares complejos.

Su objetivo principal es ofrecer una "supercomputadora virtual" capaz de procesar múltiples réplicas de simulaciones a nivel atómico. Dentro de GPUGrid, uno de los campos con mayor impacto es la simulación molecular orientada al diseño de fármacos y el análisis de interacciones proteína-ligando [27].

2.4. Plataformas existentes relacionadas con la Simulación Cardiaca

La simulación cardiaca se ha convertido en un elemento fundamental de la electrofisiología computacional, ya que permite analizar con detalle la propagación del impulso eléctrico en modelos que abarcan desde la célula aislada (como DENIS Myocyte) hasta el corazón completo. En la actualidad existen diversas plataformas diseñadas para abordar estas simulaciones a diferentes escalas y niveles de complejidad, haciendo uso de enfoques y metodologías de alto rendimiento (HPC). A continuación, se revisan tres de las más representativas: EMOS (Elvira), CHASTE y openCARP.

2.4.1. El simulador EMOS (Elvira): fundamentos, avances y proyección futura

El simulador EMOS surge como una iniciativa para abordar la resolución eficiente de las ecuaciones de la electrofisiología cardiaca, concretamente el modelo monodominio, en el que se describe la propagación del potencial transmembrana acoplado a un conjunto de ecuaciones de alta no linealidad. Desde sus primeras fases de desarrollo, este software se planteó para responder a la necesidad de resolver problemas de gran escala en tiempos razonables, dando soporte al estudio de la activación eléctrica del corazón y contribuyendo a una mejor comprensión de las arritmias y otros fenómenos de gran interés biomédico [29].

Su diseño hace énfasis en la computación de altas prestaciones (HPC). Para ello, el código implementa una estrategia de descomposición de dominio y recurre a bibliotecas paralelas que facilitan la distribución de la carga de trabajo dentro de un *cluster* de supercomputación. Esto lo convierte en una herramienta adecuada para simulaciones que abarcan grandes porciones de tejido cardiaco o incluso corazones completos [29].

2.4.2. CHASTE: un marco test-driven para la simulación fiable y escalable

El simulador CHASTE surge como una iniciativa para abordar la simulación y el análisis de la electrofisiología cardiaca de una forma altamente estructurada y verificada, situándose dentro del marco de la computación científica de altas prestaciones y del desarrollo de software con sólidas garantías de calidad [31]. A diferencia de enfoques más tradicionales, este se concibe desde sus inicios como un entorno orientado no solo a la resolución numérica de modelos (como podría ser el caso de EMOS [29]), sino también a la incorporación de prácticas de verificación y validación del código, así como al uso de metodologías de desarrollo test-driven que contribuyen a la fiabilidad de los resultados.

Una de sus características principales es su capacidad de resolver ecuaciones en varios niveles de detalle: desde modelos celulares monodominio hasta las ecuaciones más completas de la electrofisiología cardiaca, como el bidominio [31].

Desde el punto de vista del desarrollo, CHASTE adopta una metodología basada en pruebas (test-driven development), que promueve la escritura de conjuntos de test unitarios exhaustivos antes de la implementación final de las nuevas funcionalidades [31]. Este enfoque incrementa la trazabilidad y reduce la posibilidad de errores ocultos, favoreciendo la reproducibilidad de los experimentos. Para entornos donde la precisión y la validación son críticas (como en la electrofisiología cardiaca), esta estrategia se ha revelado esencial a la hora de garantizar resultados coherentes en simulaciones a gran escala.

2.4.3. openCARP: un entorno abierto para la simulación integral y multiescala en electrofisiología cardiaca

El entorno de simulación de código abierto openCARP es un entorno diseñado específicamente para estudios avanzados en electrofisiología cardiaca, que ha surgido para atender la creciente demanda de herramientas de modelado computacional accesibles y de alto rendimiento en la comunidad científica y médica. A diferencia de desarrollos previos más centrados en experimentos aislados, openCARP propone una plataforma unificada que integra simulaciones desde el nivel celular hasta el nivel del órgano completo, permitiendo abordar desde fenómenos fundamentales hasta aplicaciones clínicas de diagnóstico y terapia [33].

Este entorno de simulación se compone principalmente de dos herramientas complementarias: el simulador openCARP, desarrollado en C++ con soporte para entornos computacionales heterogéneos, y la plataforma carputils, basada en Python, que facilita la creación y gestión de experimentos computacionales complejos. El simulador destaca por la flexibilidad y facilidad de uso, permitiendo a los investigadores realizar simulaciones a distintos niveles de complejidad sin necesidad de poseer conocimientos profundos sobre aspectos numéricos o computacionales. Esta característica contribuye significativamente a reducir la curva de aprendizaje para investigadores cuyo interés primordial reside en el estudio fisiológico y clínico de problemas cardiacos, y no necesariamente en el desarrollo de código o algoritmos numéricos [33].

Una de sus fortalezas radica en su capacidad para abordar simulaciones multiescala, desde modelos celulares hasta geometrías tridimensionales complejas del corazón completo, incluyendo la integración de estructuras anatómicas detalladas derivadas de técnicas de imagen avanzada como la tomografía. Además, permite la personalización funcional de los modelos utilizando datos clínicos específicos, lo que resulta fundamental para realizar estudios de precisión y experimentos de medicina personalizada. En este sentido, openCARP ofrece flujos de trabajo estandarizados y

reproducibles, integrando formatos abiertos como CellML para modelos celulares y VTK para visualización, lo que facilita la interoperabilidad con otras herramientas y favorece la estandarización y transparencia en la investigación [33].

El simulador emplea métodos numéricos avanzados para la solución de las ecuaciones fundamentales de la electrofisiología cardiaca, como el modelo monodominio y bidominio, implementando técnicas como elementos finitos para la discretización espacial y esquemas explícitos e implícitos para la integración temporal. Estas técnicas permiten a openCARP alcanzar un alto rendimiento computacional, adecuado tanto para estaciones de trabajo locales como para infraestructuras de computación de alto rendimiento (HPC), lo que lo convierte en una herramienta especialmente útil para simulaciones de gran escala o estudios poblacionales [33].

Este también destaca por su enfoque en la calidad del software, asegurando que la implementación sea robusta y precisa mediante sistemas automatizados de pruebas continuas y control de calidad del código. La plataforma proporciona documentación detallada, tutoriales y recursos educativos accesibles que facilitan su adopción por parte de nuevos usuarios y fomentan la formación de una comunidad científica activa alrededor del proyecto. Además, cuenta con mecanismos claros para la documentación y publicación de resultados, lo que promueve una mayor reproducibilidad y confianza en los resultados obtenidos [33].

3. Objetivos

El objetivo principal del proyecto es dar un primer paso para adaptar la plataforma DENIS a la simulación multiescala y permitir así la ejecución de fibras heterogéneas, avanzando hacia las capacidades de los simuladores analizados en el estado del arte. Para ello se establecen los siguientes objetivos secundarios:

- **Desarrollar un nuevo simulador para DENIS que resuelva simulaciones 1D**: en una primera aproximación, estas células deben ser todas del mismo tipo y utilizar el mismo modelo.
- **Simulación heterogénea**: dentro de las fibras a simular, se podrán modificar los diferentes parámetros del modelo para todas las células (interpaciente) o entre las diferentes células de la simulación (intrapaciente).
- **Post-procesado en los ordenadores voluntarios**: Para reducir el flujo de información y evitar que el servidor se sobrecargue de información, al igual que en el simulador 0D, todos aquellos marcadores que sea posible se calcularán antes de que las simulaciones se devuelvan. Se deberían mantener al menos los actuales: APD90, APD50, Triangulación, etc.
- **Simulador multiplataforma**: El proyecto debe seguir siendo multiplataforma y tiene que poder compilarse el mismo código (o con pequeñas variaciones) en Windows, Linux y MacOS. En la medida de lo posible, se automatizará esta tarea.

4. Metodología

El éxito de un proyecto no depende solamente de la calidad de su implementación, sino también de la forma en que se organiza y gestiona su desarrollo. La planificación y la adopción de metodologías estructuradas permiten optimizar recursos, reducir riesgos y garantizar que el producto final cumpla con los objetivos establecidos. En este sentido, trabajar sin una metodología definida incrementa la probabilidad de desorden, retrasos y dificultades en la resolución de problemas, y puede incluso provocar el fracaso del proyecto.

Para evitar estas complicaciones, este proyecto se desarrollará siguiendo la metodología ágil SCRUM, asegurando una evolución progresiva mediante iteraciones controladas y revisiones continuas. Además, adicionalmente para la parte más técnica, se empleará GitFlow como estrategia de gestión de versiones, lo que facilitará la organización del código y la integración eficiente de nuevas funcionalidades sin comprometer la estabilidad del sistema.

4.1. Metodología SCRUM

SCRUM [34] es un marco de trabajo ligero y ágil que ayuda a personas, equipos y organizaciones a generar valor a través de soluciones adaptativas para problemas complejos. Su estructura permite un desarrollo incremental y adaptativo del proyecto mediante ciclos de trabajo llamados *Sprints*, los cuales incluyen eventos específicos para inspeccionar y adaptar tanto los artefactos producidos como el proceso en sí mismo.

En SCRUM, el trabajo se organiza en ciclos iterativos llamados *Sprints*, donde el equipo SCRUM convierte una selección de tareas en incrementos funcionales del producto. Cada *Sprint* incluye eventos específicos para inspeccionar y adaptar el producto y el proceso, tales como la planificación del *Sprint* (*Sprint Planning*), reuniones diarias de seguimiento (*Daily Scrum*), revisión del *Sprint* (*Sprint Review*) y retrospectiva del *Sprint* (*Sprint Retrospective*). Estos eventos son fundamentales para garantizar la transparencia, la inspección frecuente y la adaptación continua del trabajo realizado por el equipo.

Sus valores fundamentales son Compromiso, Foco, Apertura, Respeto y Coraje. Estos valores proporcionan al equipo la dirección necesaria para llevar a cabo su trabajo, sus acciones y su comportamiento, fortaleciendo así la confianza y asegurando que las decisiones adoptadas sean coherentes con los objetivos del proyecto.

El equipo está compuesto por un *Product Owner*, un *Scrum Master* y un grupo de *Developers*. El *Product Owner* es responsable de gestionar el *Product Backlog* y maximizar el valor del producto, el *Scrum Master* se encarga de facilitar y asegurar la correcta implementación de SCRUM, y los

Developers son los encargados de crear valor añadido al proyecto, manteniendo siempre la calidad según la *Definition of Done* establecida por el equipo.

4.2. Metodología GitFlow

GitFlow [35] es un modelo exitoso de ramificación para sistemas de control de versiones basado en Git, que busca aportar claridad y organización al proceso de desarrollo de software. Su principal objetivo es facilitar el trabajo colaborativo y gestionar eficazmente la evolución del código mediante una estructura definida y sistemática de ramas, asegurando que el código siempre se mantenga ordenado y estable.

El modelo GitFlow define dos ramas principales y permanentes: *main* (también llamada a veces *master*) y *develop*. La rama *main* alberga únicamente el código estable y probado, listo para desplegarse en entornos de producción. Por otro lado, la rama *develop* funciona como el eje central de trabajo donde se integran continuamente nuevas funcionalidades, sirviendo como base para las futuras versiones del producto.

Además de estas dos ramas centrales, GitFlow utiliza tres tipos de ramas temporales destinadas a tareas específicas: las ramas de características (*feature branches*), las ramas de lanzamiento (*release branches*) y las ramas de corrección rápida (*hotfix branches*). Las ramas de características se crean a partir de la rama *develop* para desarrollar nuevas funcionalidades, y una vez terminadas se vuelven a fusionar en *develop*. Las ramas de lanzamiento se derivan de *develop* cuando se decide preparar una nueva versión del producto, permitiendo realizar ajustes menores antes de fusionarlas tanto en *main* como en *develop*. Finalmente, las ramas de corrección rápida se utilizan para solucionar problemas críticos que surjan en producción; estas ramas se crean desde main y posteriormente se integran tanto en *main* como en *develop* para mantener la coherencia del código en todas las ramas.

La principal ventaja del uso de GitFlow reside en su capacidad para gestionar con eficacia desarrollos paralelos, evitando conflictos entre versiones y proporcionando una clara separación entre el código que se encuentra en desarrollo y el código listo para producción. Además, su estructura facilita realizar correcciones rápidas en producción sin afectar negativamente al desarrollo continuo de nuevas funcionalidades.

4.3. Justificación de las Metodologías seleccionadas

La metodología SCRUM ha sido elegida para el desarrollo de este proyecto debido a las siguientes razones principales:

- **Flexibilidad y adaptación**: El proyecto abordado presenta desafíos técnicos y conceptuales que pueden variar a medida que avanza el desarrollo. SCRUM ofrece la

capacidad de adaptarse rápidamente a cambios en los requerimientos, facilitando ajustes continuos durante cada iteración y garantizando que el proyecto siempre esté alineado con sus objetivos iniciales.

- **Colaboración efectiva**: Este trabajo requiere la interacción frecuente con el director del proyecto para asegurar que las decisiones sean coherentes con los objetivos definidos. SCRUM promueve la comunicación constante entre todos los implicados, permitiendo detectar y solucionar problemas desde etapas tempranas.
- **Iteraciones incrementales**: SCRUM propone ciclos cortos y bien definidos (*Sprints*) en los que se entregan versiones funcionales del producto de forma periódica. Esto permite evaluar frecuentemente los avances, realizar ajustes oportunos, y asegurar que cada incremento del producto cumple con los estándares de calidad definidos, disminuyendo así el riesgo de fallos acumulados al final del desarrollo.

Por otro lado, la elección de GitFlow como metodología de control de versiones se justifica principalmente debido a:

- **Gestión ordenada del desarrollo**: la metodología permite gestionar eficazmente múltiples líneas de desarrollo a través de ramas claramente definidas. Esto facilita enormemente la organización del código y garantiza que el trabajo simultáneo en diferentes funcionalidades no provoque conflictos o pérdidas de estabilidad en el producto final.
- **Respuesta ágil ante incidencias**: Gracias a su estructura organizada, esta permite solucionar rápidamente errores críticos que puedan surgir en la rama de producción. Esto es especialmente relevante para asegurar que los errores detectados en producción puedan ser resueltos inmediatamente sin afectar negativamente al desarrollo continuo de nuevas funcionalidades.
- **Claridad en la gestión de versiones**: Dado que este proyecto requerirá la generación y gestión clara de diversas versiones del producto, la estructura propuesta por GitFlow es especialmente útil. Su esquema de ramas asegura una separación clara y sencilla entre el código en desarrollo activo y las versiones estables y listas para ser desplegadas, reduciendo considerablemente la complejidad en la gestión del proyecto.

4.4. Adaptación de la Metodología de SCRUM para el Proyecto

Para llevar a cabo este proyecto de forma efectiva, SCRUM se adaptará considerando tanto las características específicas del trabajo como el contexto académico en el que se desarrolla.

4.4.1. Roles definidos en el proyecto

Dado que el proyecto será realizado individualmente, los roles tradicionales de SCRUM se han adaptado de la siguiente manera:

- **Product Owner**: Este rol lo ejerceré yo principalmente, Iván Royo Gutiérrez, mediante el cual proporcionaré las directrices iniciales y priorizaré las tareas más relevantes para la consecución de los objetivos del proyecto.
- **Scrum Master**: En este proyecto será desempeñado por el director del proyecto, Jesús Carro Fernández, quien velará por mantener la aplicación correcta del marco SCRUM y solucionará cualquier impedimento y/o bloqueo que pueda llegar a surgir.
- **Equipo de Desarrollo (*Developers*)**: Este rol será igualmente asumido por mí, Iván Royo Gutiérrez, en el cual seré responsable de la planificación, desarrollo, pruebas e implementación del producto final.

4.4.2. Eventos y estructura de los Sprints

El proyecto se dividirá en ciclos iterativos llamados *Sprints*, cada uno con una duración estimada de una a cuatro semanas. Esta duración permitirá realizar un seguimiento continuo y corregir posibles desviaciones rápidamente. Cada *Sprint* incluirá:

- **Sprint Planning**: Al comienzo de cada *Sprint*, se seleccionarán y definirán claramente las tareas a desarrollar, teniendo en cuenta la priorización dada por el *Product Owner* y las necesidades inmediatas del proyecto.
- **Sprint Review**: Al finalizar cada *Sprint*, se realizará una revisión con el tutor, donde se analizarán los resultados alcanzados, proporcionando así retroalimentación inmediata para mejorar la calidad del trabajo realizado.
- **Sprint Retrospective**: Al finalizar cada iteración, se realizará un breve análisis sobre el desempeño durante el *Sprint*, identificando aspectos a mejorar en el siguiente ciclo y garantizando así una mejora continua en el desarrollo del proyecto.

Debido a la naturaleza individual del proyecto y al marco académico en el que se desarrolla el Trabajo de Fin de Grado, no resulta viable realizar reuniones diarias de seguimiento, ya que no existe un equipo físico con quien comunicarse diariamente, es por eso por lo que para este proyecto aboliremos el *Daily Scrum*.

4.5. Adaptación de la metodología GitFlow para el proyecto

Esta adaptación se centra en la integración con la plataforma de computación voluntaria BOINC utilizada en el proyecto.

BOINC permite diferenciar claramente entre versiones definitivas y versiones beta mediante el archivo *version.xml*. Este archivo, que indica el nombre del binario de la aplicación que ejecutarán los voluntarios, admite añadir un tag *<beta/>* para distinguir versiones en fase de pruebas.

Aprovechando esta característica de BOINC, la metodología GitFlow del proyecto se adaptará en la nomenclatura de versiones del repositorio, siguiendo dos patrones claros y sistemáticos:

- **Versiones definitivas**: se publicarán siguiendo un esquema tradicional de numeración incremental, por ejemplo, 0.01, 0.02, etc. Estas versiones estarán completamente validadas y listas para despliegue en entornos productivos.
- **Versiones beta**: destinadas exclusivamente para pruebas y ajustes antes del lanzamiento definitivo, usarán un esquema especial con el prefijo *beta*, como por ejemplo *beta_0.01*. Esto permitirá mantener claras las diferencias y simplificar la gestión y distribución de versiones experimentales entre los voluntarios del proyecto.

De esta manera, la adaptación propuesta se limita a una integración sencilla y efectiva del sistema de etiquetado propio de BOINC con la estructura de ramas temporales propuesta por GitFlow, facilitando la distinción entre versiones de prueba y versiones listas para producción.

4.6. Planificación Inicial del Proyecto

Con el objetivo de establecer una planificación clara y realista, y de mantener el desarrollo del proyecto dentro de un marco temporal establecido, se ha realizado inicialmente una estimación de las tareas necesarias para la consecución del proyecto, tanto a nivel tecnológico como documental. A continuación, se muestra el Diagrama de Gantt generado a partir de esta planificación inicial, en el que quedan especificadas las fechas previstas para cada una de las tareas. Dada la metodología empleada, esta planificación sufrirá alteraciones, pero servirá de marco de referencia para velar por la correcta evolución del producto.

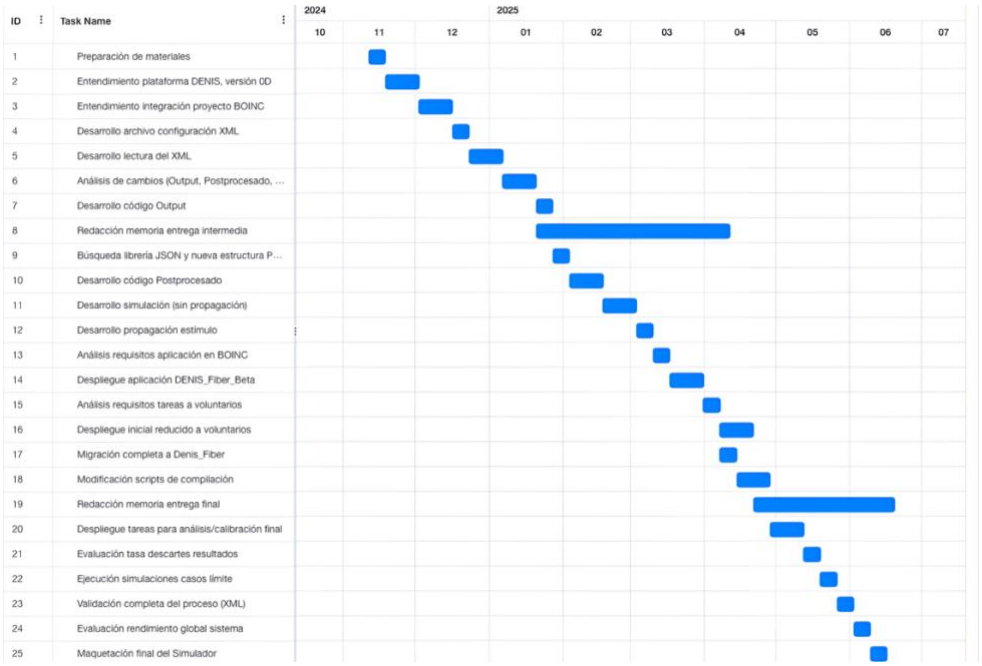


Figura 8: Diagrama de Gantt sobre la Planificación Inicial

Para facilitar la gestión efectiva de estas tareas, se utilizará la plataforma Trello, una herramienta online que permite la organización y seguimiento visual de actividades mediante tableros y listas. Esta plataforma se adapta especialmente bien a la metodología SCRUM definida previamente, ya que permite realizar un seguimiento ágil de los distintos sprints definidos en el proyecto, mostrando claramente el estado de cada tarea.

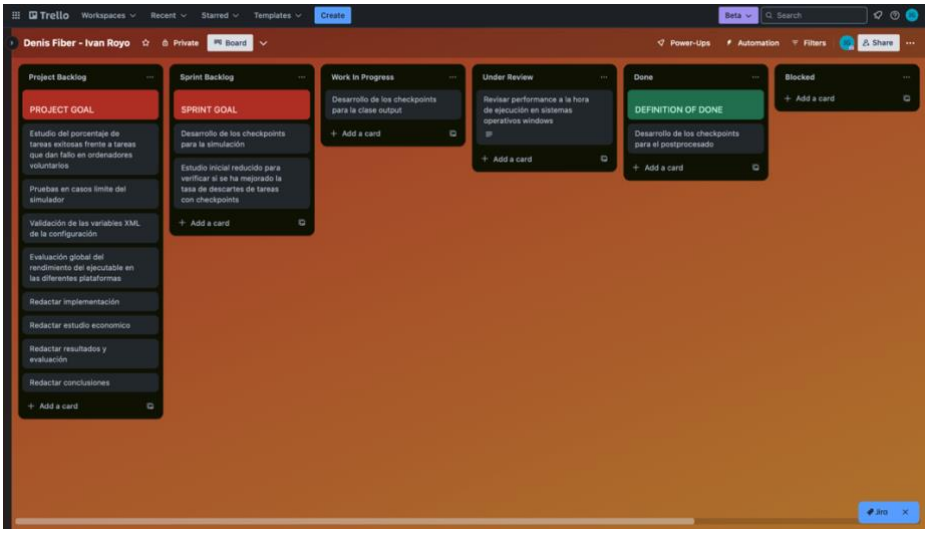


Figura 9: Tablero del proyecto para la aplicación de SCRUM

Gracias a la flexibilidad y claridad visual que ofrece Trello, se podrá mantener un control exhaustivo del desarrollo del proyecto, identificar rápidamente posibles desviaciones en la planificación inicial y asegurar el cumplimiento de los plazos establecidos.

5. Análisis e Implementación

La calidad de un proyecto de ingeniería no se juzga únicamente por los resultados, sino por la solidez con que se materializan sus componentes y se garantiza su operatividad en los distintos entornos de uso. Una implementación planificada, documentada y reproducible es clave para la fiabilidad científica, la escalabilidad futura y la transferencia tecnológica. Trabajar sin un proceso de implementación definido conllevaría inconsistencias entre plataformas, dificultades de mantenimiento y, en última instancia, la pérdida de confianza en los resultados obtenidos.

Con estos principios en mente, este capítulo expone el camino seguido para convertir los requisitos del proyecto en un sistema funcional. Se presentará, en primer lugar, el simulador DENIS Fiber, núcleo computacional desarrollado en C++; se analizará a continuación su integración en la infraestructura BOINC de DENIS@Home, incluyendo así la nueva aplicación a este proyecto; se describirá el entorno de pruebas reproducible basado en Docker-Compose, empleado para validar cada iteración antes del despliegue; y se detallará, por último, la estrategia de compilación y empaquetado multiplataforma, que automatiza la entrega de binarios para Windows, Linux y MacOS.

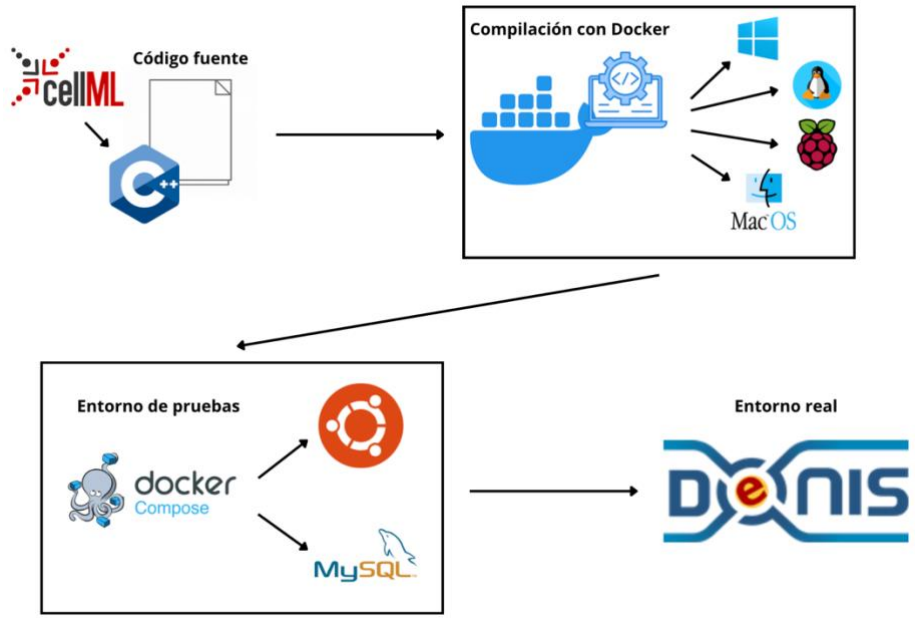


Figura 10: Vista general de la implementación del sistema completo.

5.1. Análisis del problema de la propagación del impulso eléctrico en una fibra cardiaca

La propagación del impulso eléctrico dentro de una fibra se modela mediante una ecuación de difusión, derivada del modelo de monodominio ampliamente utilizado en electrofisiología

cardiaca. La ecuación diferencial que calcula la propagación de la actividad eléctrica utilizada en este simulador tiene la siguiente forma general, para toda posición e instante de tiempo [36]:

$$C_m \frac{\partial V(x,t)}{\partial t} = -(I_{ion}(x,t) - I_{stim}(x,t)) + \sigma \frac{\partial^2 V(x,t)}{\partial x^2}$$

Donde C_m es la capacitancia de la membrana, V es el potencial de la membrana, I_{ion} es la suma de todas las corrientes iónicas transmembrana, I_{stim} es la corriente de estimulación aplicada externamente, y σ es el coeficiente de difusión del tejido.

El coeficiente de difusión depende del tejido que se quiera simular y del modelo que se esté aplicando. Este valor se ajusta para tener una velocidad de conducción fisiológica. Por ejemplo, para los cálculos de propagación con el modelo TP06, se suele establecer un valor de $\sigma = 0{,}0012102 \frac{cm^2}{ms}$, tal que produzca una velocidad de conducción máxima de $74\ cm/s$ [37].

En electrofisiología celular, es habitual trabajar con la capacitancia de la membrana normalizada, por lo que $C_m = 1\mu F/cm^2$ [36], de modo que, al escoger unidades de corriente en $\mu A/cm^2$, de potencial en mV y de tiempo en ms, la constante C_m pasa a valor adimensional unidad. Esto equivale a reescalar la variable temporal (absorbiendo C_m en la definición de t) sin alterar la dinámica del sistema, y permite simplificar la ecuación, para toda posición, a:

$$\frac{\partial V(x,t)}{\partial t} = -(I_{ion}(x,t) - I_{stim}(x,t)) + \sigma \frac{\partial^2 V(x,t)}{\partial x^2}$$

Para resolver numéricamente la ecuación de propagación, se decidió utilizar un esquema explícito basado en Métodos de Diferencias Finitas (FDM), y en particular, aplicando el método de las líneas en la dimensión espacial. Para ello, es necesario discretizar el término difusivo espacial $\frac{\partial^2 V}{\partial x^2}$ utilizando el esquema clásico de diferencias finitas centradas de segundo orden [36]:

$$\frac{\partial^2 V(x_i,t)}{\partial x^2} \approx \frac{V_{i-1} - 2V_i + V_{i+1}}{\Delta x^2}$$

Donde Δx es el paso de discretización espacial, i es el índice de la posición en la fibra (de 0 a N), x_i es la posición en la fibra ($x_i = \Delta x \cdot i$), y V_i es el potencial de la célula de la posición x_i. Este método es ampliamente aceptado en la literatura para resolver ecuaciones de tipo monodominio en una dimensión, convirtiendo un problema de dos dimensiones, en un sistema de ecuaciones ordinarias acopladas de una sola dimensión. Al final, si substituimos, obtendríamos la siguiente formula:

$$\frac{\partial V_i}{\partial t} \approx -(I_{ion} - I_{stim}) + \sigma \frac{V_{i-1} - 2V_i + V_{i+1}}{\Delta x^2}$$

Este tipo de problemas, para estar bien definidos, necesitan tener condiciones iniciales (que en nuestro caso vendrán definidas en el propio modelo a simular), y condiciones de borde a la hora de calcular el termino difusivo espacial ($\frac{\partial^2 V}{\partial x^2}$). Estas condiciones de borde nos servirán para calcular el potencial en la izquierda y en la derecha, ya que en esas posiciones no diponemos de valores más a la izquierda o más a la derecha respectivamente. Para estos extremos, imponemos condiciones de contorno de tipo Neumann homogéneas (o "sealed-end"), que establecen que el flujo de corriente en los extremos de la fibra debe ser nulo, es decir:

$$\left.\frac{\partial V}{\partial x}\right|_i = 0$$

Si aproximamos la derivada con diferencias finitas centradas, nos queda que:

$$\left.\frac{\partial V}{\partial x}\right|_i \approx \frac{V_{i+1} - V_{i-1}}{2\Delta x}$$

Por lo tanto, si exigimos que esto sea cero, obtendremos que:

$$\frac{V_{i+1} - V_{i-1}}{2\Delta x} = 0 \implies V_{i+1} - V_{i-1} = 0 \implies V_{i+1} = V_{i-1}$$

Al final, si aplicamos estas condiciones de contorno, la ecuación podría expresarse de la siguiente forma:

$$\frac{\partial^2 V_i}{\partial x^2} \approx \begin{cases} \frac{2V_1 - 2V_0}{\Delta x^2} \; Para\; x_i = 0 \\ \frac{V_{i-1} - 2V_i + V_{i+1}}{\Delta x^2} \; Para\; 0 < x_i < N \\ \frac{2V_{l-2} - 2V_{l-1}}{\Delta x^2} \; Para\; x_i = l \end{cases}$$

Siendo l el número de posiciones en las que se ha discretizado la fibra cardiaca, e x_i la posición en la fibra para la que se está tratando de calcular el voltaje. De este modo, la "célula fantasma" en la posición x_{i+1} adopta el mismo valor que su reflejo interior x_{i-1}, garantizando el flujo cero en los extremos.

En este punto, se conectaría con el proyecto unicelular DENIS Myocyte cuyo el solver ya incluye el cálculo del voltaje para una célula aislada utilizando diferentes métodos de resolución explícitos, que, en este caso, sería [36]:

$$\frac{\partial V_m}{\partial t} = -(I_{ion} - I_{stim})$$

5.2. Simulador Denis Fiber

El simulador DENIS Fiber agrupa en una única aplicación todos los componentes necesarios para llevar a cabo simulaciones cardiacas distribuidas. En su núcleo se encuentran dos librerías internas:

- **denis_fiber_lib**, que implementa la lógica principal de la simulación (lectura de configuración, métodos de resolución de ecuaciones diferenciales, gestión del bucle temporal, checkpoints, etc.).
- **models_lib**, que contiene los modelos celulares preparados (derivados de archivos CellML convertidos a cabeceras de C) y proporciona las funciones para inicializar estados, calcular tasas y obtener nombres de variables.

Además, el ejecutable principal (*main*) se ha adaptado para funcionar sobre la infraestructura BOINC, delegando en la librería BOINC y en su API (boinc_api) todas las tareas relacionadas con comunicación con el servidor, gestión de archivos de entrada/salida y control de progreso en los clientes voluntarios. Finalmente, la biblioteca jsoncpp se integra para manejar la generación y escritura del archivo de postprocesado en formato JSON. Todas estas librerías externas se compilan previamente antes de enlazarse con el proyecto.

5.2.1. Archivo de configuración de la simulación

El punto de entrada de cualquier ejecución del simulador es un fichero XML que describe, de forma declarativa, todos los parámetros necesarios para generar y resolver la fibra cardiaca. El esquema parte del formato empleado en la versión 0D (DENIS Myocyte) pero introduce nuevas secciones que permiten: (i) tratar dominios unidimensionales, (ii) definir estímulos locales y (iii) acotar la información que se desea registrar o postprocesar.

A grandes rasgos, podemos dividir el fichero en cuatro bloques jerárquicos. A continuación, se detalla cada uno de ellos.

Configuración general

Inmediatamente bajo la raíz se declara la naturaleza del experimento con `<type>`, que en este caso será fiber (simulación unidimensional). El elemento `<model>` indica el nombre del modelo CellML que se usará para las células de la simulación; de él se extraerán variables, constantes y ecuaciones.

El potencial que actuará como variable principal, y por tanto será en la que se apliquen los cálculos para la propagación del estímulo, se identifica dentro de `<potential>` mediante `<name>` y `<component>`, de modo que el simulador localice el tipo de potencial que estamos especificando dentro de la estructura de datos generada por CellML.

Los parámetros <time> y <dt> fijan, respectivamente, la duración total y el paso temporal (ambos en las mismas unidades en las que el tiempo esté definido en el fichero CellML). Por último, <solver> selecciona el método numérico que se utilizará para resolver las ecuaciones diferenciales en el tiempo (FE para Forward Euler, RK2 para Runge-Kutta de orden 2, RK4 para Runge-Kutta de orden 4, etc.). La resolución de las ecuaciones diferenciales de la propagación del potencial eléctrico se realiza, como se ha indicado anteriormente, mediante diferencias finitas y el Método de las Líneas utilizando una aproximación espacial de segundo orden y un esquema explicito en el tiempo.

Bloque de la fibra

El elemento <fiber> introduce los parámetros geométricos y de estimulación específicos de la fibra:

- <length> define la longitud física en cm;
- <dx> establece el espaciado entre las diferentes posiciones que se considerarán para realizar la integración. A partir de este dato se puede calcular el número de segmentos en los que se divide la fibra (N = length / dx).
- <diffusion_coefficient> establece el coeficiente de difusión σ el cual se comenta más adelante en el apartado 5.2.2.

Dentro de este, también podemos encontrar <positions>, donde es posible sobrescribir constantes del modelo para posiciones concretas. Cada <position index="n"> señala la posición n (0-basado) y contiene un único <constants_to_change>. El simulador hereda la estimulación de la célula que se haga dentro del CellML en el que está descrito el modelo, por lo que se debe utilizar esta parte para adaptar en qué célula se estimula y con qué parámetros. Cada <constant> subyacente detalla <name>, <component> y <value>; cualquier constante de CellML es susceptible de modificación.

También podemos especificar el elemento <global_constants_to_change>, el cual permite modificar simultáneamente constantes específicas para todas las posiciones dentro de la fibra. Esto resulta especialmente útil cuando se desea aplicar cambios generales al modelo de células sin necesidad de definirlo explícitamente posición por posición. La estructura de este elemento es idéntica al anterior, especificando <name>, <component> y <value> para cada constante global a modificar. En caso de que se modifique una constante en todas las posiciones con <global_constants_to_change> y en una posición en concreto con <constants_to_change>, prevalece en esa posición el valor indicado en <constants_to_change>.

Configuración de salida

El bloque `<output>` controla qué se almacena durante la simulación. El atributo id lista las células observadas (índices separados por comas: "*0,1,2,3,4*" o un rango de números representado de la siguiente manera: "*0:10*" equivalente a desde el 0 hasta el 10). Las sub-etiquetas `<start>` y `<freq>` definen, en las unidades de tiempo del modelo y en número de pasos respectivamente, el momento a partir del cual comienzan las capturas y cada cuántos pasos se repite la operación.

El usuario puede seleccionar las magnitudes que desea registrar, contenidas dentro del modelo que se use:

- `<states>` agrupa variables de estado (p. ej., V, m, h).
- `<algebraics>` permite incluir los valores de variables algebraicas del modelo (I_Na, etc.).
- `<rates>` hace lo propio con derivadas de las diferentes variables de estado (dV/dt, dm/dt, …).

Postprocesado

Finalmente, `<postprocessing>` especifica una lista de posiciones (id="" al igual que en output) y un instante `<start>` a partir del cual el simulador calculará, una vez concluida la integración, y a partir del voltaje escogido como variable de potencial, marcadores electrofisiológicos como APD90, Triangulación o Voltaje Máximo. Estos resultados se devuelven en un archivo de formato JSON (se detalla más adelante en el punto 5.2.4).

La lectura y procesamiento del archivo XML descrito se lleva a cabo mediante la clase *DenisConfig*, que actúa como intermediaria entre la estructura jerárquica del fichero y las variables internas del simulador. Al iniciar la simulación, esta clase interpreta todos los bloques anteriormente descritos y traduce su contenido a estructuras de datos adecuadas, como vectores de variables, identificadores de posiciones o listas de constantes a modificar. Esto permite que el simulador acceda a toda la información de configuración durante la ejecución.

A continuación, se puede ver un ejemplo de una configuración de una simulación sencilla, de un segundo de duración, que usa el modelo de ten Tusscher Panfilov (TP06) [18]:

```xml
<?xml version="1.0"?>
<simulation>
    <!-- Configuracion general -->
    <type>fiber</type> <!-- Tipo de simulacion -->
    <model>TP06_ENDO_SS_biphasic</model>
    <potential>
        <name>V</name>
        <component>membrane</component>
    </potential>
    <time>1000</time>
    <dt>0.0015</dt>
    <solver>FE</solver>

    <!-- Parametros especificos de la fibra -->
    <fiber>
        <length>3</length> <!-- Longitud en centimetros -->
        <dx>0.01</dx> <!-- Espaciado en centimetros -->
        <diffusion_coefficient>0.0012102</diffusion_coefficient> <!-- Coeficiente de difusion -->
        <!-- Constantes a cambiar para toda la fibra -->
        <global_constants_to_change>
            <constant>
                <name>G_Kr</name>
                <component>rapid_time_dependent_potassium_current</component>
                <value>0.213</value>
            </constant>
        </global_constants_to_change>
        <!-- Constantes a cambiar para cada posicion -->
        <positions>
            <position index="0">
                <constants_to_change>
                    <constant>
                        <name>I_Stim_Amplitude</name>
                        <component>membrane</component>
                        <value>-312.0</value>
                    </constant>
                </constants_to_change>
            </position>
        </positions>
    </fiber>

    <!-- Configuracion de salida -->
    <output id="125,150,175,200,225"> <!-- Ids de las posiciones a guardar -->
        <start>0</start>
        <freq>50</freq>
        <states>
            <state>
                <name>V</name>
                <component>membrane</component>
            </state>
        </states>
    </output>

    <!-- Configuracion de postprocesado -->
    <postprocessing id="125,150,175,200,225">
        <start>0</start>
    </postprocessing>
</simulation>
```

Figura 11: Ejemplo archivo básico de configuración XML

5.2.2. Simulación de una fibra cardiaca

Durante la simulación de una fibra, tienen lugar varios procesos dentro del simulador, que básicamente se dividirían en dos secciones: la creación de la estructura de la simulación, y el bucle de la simulación.

Creación de la estructura de la simulación

Antes de iniciar la simulación, es necesario construir y preparar todas las estructuras internas que permitirán realizar los cálculos numéricos durante la ejecución del simulador.

La inicialización del simulador se basa en dos estructuras fundamentales: por un lado, la configuración previamente leída desde el archivo XML (detallada en el apartado anterior), que define todos los parámetros de la simulación; y, por otro lado, la conexión a BOINC (BoincConnection, detallada más adelante en el apartado 5.2.5), cuya tarea principal es gestionar las comunicaciones con la API de BOINC y proveer rutas y mecanismos para interactuar con archivos y checkpoints.

El constructor de la clase *Simulation* es el encargado de inicializar todas las variables y estructuras necesarias para la simulación. En primer lugar, se crean las estructuras que almacenarán los estados (*states*), las tasas (*rates*) y las variables algebraicas (*algebraic*) para cada una de las posiciones que componen la fibra cardiaca, así como las estructuras que contendrán los métodos de resolucion (*solvers*) y los modelos, cuyo número dependerá del número de posiciones en los que se discretizará la fibra.

Una vez inicializadas estas estructuras básicas, se procede a cargar el modelo celular especificado en la configuración para cada posición dentro de la fibra (*prepareModels()*). Debido a que se trata de una fibra heterogénea en sus características pero homogéneo en el tipo de células, cada posición cargará inicialmente el mismo modelo celular. Posteriormente, se realizan modificaciones específicas en las constantes del modelo, tanto para aquellas posiciones definidas específicamente (intrapaciente), como para las definidas globalmente (interpaciente), lo que permite simular variaciones fisiológicas dentro del tejido cardiaco.

Después de generar dichas estructuras, se procede a obtener el índice del voltaje, el cual nos será de gran utilidad posteriormente para conocer el voltaje de cada posición y realizar cálculos posteriores.

Tras esta fase, el simulador inicializa los estados iniciales de cada posición, definidos por el propio modelo celular (*Model::getInitStates()*), junto con los vectores correspondientes de variables algebraicas y derivadas. Paralelamente, se crea un solver numérico específico para cada modelo celular (*createSolverByName()*), que se encargará durante la simulación de integrar numéricamente la evolución temporal de los estados de esa posición (aplicando el método de las líneas).

Posteriormente, el constructor define variables clave como el tiempo actual (*time*), el tiempo total de simulación (*endTime*) y el paso temporal (*dt*). Además, gracias al tiempo total de la simulación, y el paso temporal, podremos calcular el número total de iteraciones que se ejecutarán en el bucle principal, el cual es un imprescindible para el bucle posterior, que se encargará de ejecutar la simulación.

Finalmente, durante esta fase inicial se establecen también las estructuras que controlan qué posiciones específicas serán guardadas durante la ejecución (*Output*) y cuáles serán postprocesadas al finalizar (*Postprocessing*). Ambas estructuras se definen a partir de los datos proporcionados en la configuración XML, obteniendo los indicadores correspondientes a ellos en el modelo. Posteriormente se hablará como se gestionan estas clases en más detalle.

El bucle de la simulación

Una vez finalizada la fase inicial de configuración y preparación de las estructuras internas, comienza la etapa principal de la simulación: el bucle iterativo que calcula la evolución temporal de la fibra cardiaca. Este bucle es el núcleo computacional del simulador, y en cada iteración se evalúan y actualizan todas las variables del modelo de acuerdo a los parámetros especificados.

```
void Simulation::runSimulation() {
    while (iteration <= numIterations) {
        time = iteration * dt;

        if (output->checkIfIterationToSave(iteration)) {
            // Calcular rates para todas las células primero
            for (int i = 0; i < numCells; i++) {
                model[i]->computeRates(time, rates[i], states[i], algebraic[i]);
                model[i]->computeAlgebraics(time, rates[i], states[i], algebraic[i]);
            }
            output->saveIteration(iteration, states, rates, algebraic);
        }

        for(int i = 0; i < numCells; i++) {
            voltages[i] = states[i][voltageIndex]; //Actualizamos voltajes
        }

        // Actualizar estados y voltajes
        postprocessing->postProcessingResult(time, states);
        for (int i = 0; i < numCells; i++) {
            solver[i]->updateStates(time, dt, states[i]);
            double V_prev = (i == 0) ? voltages[i+1] : voltages[i - 1];
            double V_next = (i == numCells - 1) ? voltages[i-1] : voltages[i + 1];
            states[i][voltageIndex] += dt * ( sigma * ((V_prev - 2 * voltages[i] + V_next) / (dx * dx)));
        }

        iteration++;

        if (boincConnection->timeToCheckpoint()) {
            saveCheckpoint();
        }

        boincConnection->updateFractionDone(((double) (iteration - 1)) / numIterations);

    }

    fprintf(stderr, "Total time: %f\n", time);
    fprintf(stderr, "State Variables:\n");

    for (int i = 0; i < numCells; i++) {
        fprintf(stderr, "Cell %d:\n", i);
        for (int j = 0; j < model[i]->getStatesLength(); j++)
            fprintf(stderr, " - %s: %f\n", model[i]->getStateName(j), states[i][j]);
    }
}
```

Figura 12: Método encargado de abordar la simulación de la fibra cardiaca

En primer lugar, dentro del bucle principal, se actualiza el tiempo de simulación actual multiplicando el número de iteración por el paso temporal previamente definido. Esta variable temporal (*time*) se utiliza a partir de aquí como referencia para evaluar diversas condiciones y realizar cálculos en cada iteración.

Seguidamente, se verifica si la iteración actual corresponde con la iteración en la que debemos de empezar a guardar los datos de salida (*output*), según lo que se ha especificado previamente en la configuración inicial. Si así fuese, el simulador procederá a calcular las tasas (*rates*) y las

variables algebraicas (*algebraics*) para todas las posiciones de la fibra. Aunque al inicializar los modelos sólo se inicializan las variables necesarias para el cálculo básico, en esta fase el simulador necesita generar todas las variables (tanto *rates* como *algebraics*), ya que alguna de estas no es calculada en la creación inicial del modelo, si no únicamente cuando se llama a su correspondiente método; por ello, si no se calculan, podría causar problemas posteriores de haberse indicado aguardar (archivo de salida) en la configuración inicial. Tras estos cálculos, la estructura encargada del output almacena estos resultados, facilitando su posterior análisis y estudio.

A continuación, el simulador actualiza el vector de voltajes de cada célula a partir del estado actual almacenado previamente. Este paso es crítico, ya que la propagación del estímulo eléctrico a lo largo de la fibra depende directamente de estos valores actualizados del potencial de membrana.

La actualización de los estados celulares constituye la parte central del método numérico del simulador. Para ello, se hace uso del solver numérico previamente inicializado. En particular, se emplea un método explícito, típicamente Forward Euler, para resolver las ecuaciones diferenciales ordinarias del modelo. Este método explícito utiliza los valores conocidos en la iteración actual para aproximar numéricamente los valores futuros [44].

Tras actualizar numéricamente los estados internos mediante el solver, el simulador realiza el cálculo explícito de la propagación del estímulo eléctrico a lo largo de la fibra cardiaca utilizando las ecuaciones descritas en el apartado 5.1.

En primer lugar, se resuelve la ecuación diferencial para cada una de las posiciones tal y como se hacía en el simulador DENIS Myocyte, y posteriormente aplicando la parte correspondiente a la propagación espacial. Al final, si plasmamos todo esto en el código, teniendo en cuenta las condiciones de contorno, quedaría de la siguiente manera:

```
for (int i = 0; i < numCells; i++) {
    solver[i]->updateStates(time, dt, states[i]);
    double V_prev = (i == 0) ? voltages[i+1] : voltages[i - 1];
    double V_next = (i == numCells - 1) ? voltages[i-1] : voltages[i + 1];
    states[i][voltageIndex] += dt * ( sigma * ((V_prev - 2 * voltages[i] + V_next) / (dx * dx)));
}
```

Figura 13: Fragmento encargado de calcular la propagación del voltaje y actualizar los estados

Finalmente, tras completar estos cálculos, el simulador incrementa el contador de iteraciones y verifica si se debe realizar un *checkpoint* (un guardado temporal del estado de la simulación para prevenir pérdidas en caso de interrupciones, que se detallara más adelante en el punto 5.2.5). Si corresponde, se ejecuta dicho guardado. Para concluir la iteración, el simulador actualiza la fracción completada de la simulación a través de la API de BOINC, proporcionando así

retroalimentación directa al usuario sobre el progreso realizado hasta el momento en su cliente local.

Este bucle se repetirá hasta alcanzar el número total de iteraciones definido por la duración total de la simulación.

5.2.3. Gestión del output

Durante el proceso de simulación, la generación y almacenamiento de resultados intermedios es fundamental para el análisis posterior del comportamiento electrofisiológico de la fibra simulada. Por ello, es importante guardar un archivo de salida con aquellas variables de interés.

Funcionamiento de la clase Output

La gestión del output se realiza mediante una clase específica denominada *Output*, cuya principal función es controlar qué datos se almacenan durante la simulación, cuándo se almacenan y cómo se organizan estos datos para su posterior análisis.

En la fase de inicialización, la clase Output establece sus estructuras internas a partir de la información proporcionada en la configuración inicial. Específicamente, determina el instante de tiempo a partir del cual empezará a registrar resultados (*timeStart*), la frecuencia con la que se almacenarán estos datos (expresada en iteraciones), así como las posiciones concretas de la fibra cardiaca sobre las que se desea registrar información.

Una vez establecidas estas configuraciones básicas, el constructor de la clase crea las estructuras que almacenarán los resultados durante la simulación. Para cada célula seleccionada, se reservan matrices específicas que permitirán guardar los *states*, las *rates* y las *algebraics*.

Para determinar qué variables específicas guardar de cada célula, la clase Output utiliza tres funciones auxiliares (*setStatesToSaveStructures*, *setRatesToSaveStructures* y *setAlgebraicsToSaveStructures*), que comprueban qué variables definidas en la configuración existen realmente dentro del modelo celular y generan así estructuras de indexación específicas para acceder a cada variable durante la ejecución. Como resultado final, obtendremos matrices que tienen un tamaño definido por el número de posiciones y el número total de iteraciones que se van a guardar multiplicado por el número de variables especificadas para cada categoría.

Posteriormente, tras la creación de las estructuras necesarias, durante cada iteración del bucle principal de simulación, la clase Output verifica mediante el método *checkIfIterationToSave()* si la iteración actual debe guardarse, comparándola con los criterios definidos previamente. En caso afirmativo, el método *saveIteration()* registra los valores actuales de los estados, sus derivadas y las variables algebraicas de las posiciones seleccionadas en las estructuras internas previamente

mencionadas. Para ello, se usan los índices calculados a partir del número de iteración actual y las posiciones específicas de cada posición y variable configurada.

Cuando finaliza la simulación, la clase Output genera un archivo final que contiene todos los datos registrados organizados de manera estructurada y legible. A continuación, se detallará la estructura de dicho archivo.

Estructura del archivo de Output

El archivo generado por la simulación tiene una estructura tabular, que refleja directamente las variables especificadas previamente en la configuración del archivo XML. En este archivo se ha mantenido una estructura equivalente a la utilizada en el simulador DENIS Myocyte. En primer lugar, incluye una línea de cabecera que identifica cada columna del archivo. Esta línea comienza siempre con una primera columna que contiene el tiempo (*time*) correspondiente a cada iteración guardada, seguida de tantas columnas como variables (*states*, *rates* o *algebraics*) se hayan especificado para registrar, indicando además la posición específica de cada célula dentro de la fibra.

A continuación, cada una de las líneas siguientes representa un instante específico guardado durante la simulación. La primera columna de cada línea indica el tiempo exacto de la simulación, expresado en milisegundos. Seguidamente, se muestran los valores numéricos correspondientes a las variables elegidas previamente en la configuración para cada una de las células seleccionadas, manteniendo la coherencia con el orden definido en la cabecera.

```
"time in component environment (millisecond)"   "V in component membrane (millivolt) at position 125"   "V in component membrane
(millivolt) at position 150"   "V in component membrane (millivolt) at position 175"   "V in component membrane (millivolt) at
position 200"   "V in component membrane (millivolt) at position 225"
0.000000000000e+00  -8.523000000000e+01 -8.523000000000e+01 -8.523000000000e+01 -8.523000000000e+01 -8.523000000000e+01
7.500000000000e-02  -8.523687033277e+01 -8.523687033277e+01 -8.523687033277e+01 -8.523687033277e+01 -8.523687033277e+01
1.500000000000e-01  -8.524364938342e+01 -8.524364938342e+01 -8.524364938342e+01 -8.524364938342e+01 -8.524364938342e+01
2.250000000000e-01  -8.525033861894e+01 -8.525033861894e+01 -8.525033861894e+01 -8.525033861894e+01 -8.525033861894e+01
3.000000000000e-01  -8.525693941741e+01 -8.525693941741e+01 -8.525693941741e+01 -8.525693941741e+01 -8.525693941741e+01
3.750000000000e-01  -8.526345308353e+01 -8.526345308353e+01 -8.526345308353e+01 -8.526345308353e+01 -8.526345308353e+01
4.500000000000e-01  -8.526988085993e+01 -8.526988085993e+01 -8.526988085993e+01 -8.526988085993e+01 -8.526988085993e+01
5.250000000000e-01  -8.527622393670e+01 -8.527622393670e+01 -8.527622393670e+01 -8.527622393670e+01 -8.527622393670e+01
6.000000000000e-01  -8.528248345910e+01 -8.528248345910e+01 -8.528248345910e+01 -8.528248345910e+01 -8.528248345910e+01
```

Figura 14: Ejemplo de estructura de un archivo de salida (output)

5.2.4. Postprocesado en los ordenadores voluntarios

Durante la simulación, además del archivo de salida anteriormente descrito, el simulador genera también información relevante mediante el postprocesado en los ordenadores voluntarios. Este, analiza en tiempo real los resultados numéricos obtenidos por el modelo para calcular diversos marcadores electrofisiológicos útiles para el análisis posterior.

Funcionamiento de la clase encargada del postprocesado

El postprocesado se realiza mediante la clase *Postprocessing*, cuyo objetivo principal es calcular y almacenar marcadores específicos para cada potencial de acción detectado en las células que se han indicado previamente en la configuración XML. A diferencia del output convencional, el postprocesado no almacena directamente los valores brutos del potencial de membrana o derivados, sino que extrae parámetros para caracterizar cada potencial de acción generado durante la simulación.

Cuando se inicializa esta clase, se definen las estructuras internas necesarias, seleccionando las posiciones que serán analizadas según lo indicado en el archivo de configuración. Para cada posición seleccionada, la clase almacena variables internas que se utilizarán durante el proceso, tales como el potencial máximo y mínimo alcanzado durante cada potencial de acción, así como las pendientes máximas y mínimas del potencial.

Durante la ejecución de la simulación, en cada iteración, la función principal *postProcessingResult()* revisa continuamente la evolución del potencial transmembrana en cada posición seleccionada. Cuando detecta que comienza un nuevo potencial de acción (mediante la identificación de un aumento pronunciado en la pendiente del voltaje), establece un punto inicial de referencia temporal. A partir de ese momento, monitoriza constantemente las variaciones en el potencial hasta que este vuelve a descender hacia el valor basal.

La detección de los marcadores electrofisiológicos clave se realiza en la función auxiliar *updateAPMarkers()*. Esta función calcula los tiempos en los que el potencial atraviesa diferentes umbrales característicos del potencial de acción (10%, 25%, 50%, 75% y 90% de la repolarización). A partir de estas mediciones, calcula los intervalos denominados APD10, APD25, APD50, APD75 y APD90, que corresponden respectivamente a las duraciones del potencial de acción al 10%, 25%, 50%, 75% y 90% de repolarización. Además, se calcula la triangulación (diferencia entre APD90 y APD50), los voltajes máximo y mínimo alcanzados durante cada potencial de acción, así como las pendientes máximas y mínimas del potencial.

Una vez detectado el retorno del potencial por debajo del 90% de repolarización, la función auxiliar *saveAP()* guarda todos estos marcadores calculados en estructuras internas específicas, para posteriormente almacenarlos en un archivo final.

Al finalizar completamente la simulación, el método *saveToFile()* almacena estos resultados de postprocesado en un archivo estructurado en formato JSON, que facilitará su análisis posterior mediante herramientas automatizadas o software de análisis numérico.

Estructura del archivo de postprocesado

Los resultados generados durante el proceso de postprocesado se almacenan en un archivo estructurado en formato JSON, mediante la librería externa de código abierto "jsoncpp" [38]. En este caso, se ha tenido que definir una nueva estructura de fichero de marcadores diferente a la del simulador DENIS Myocyte, porque al calcular marcadores en diferentes posiciones, no está garantizado que todos ellos tengan el mismo número de eventos, por lo que la estructura tabular podría ser un problema. El archivo está compuesto por un objeto raíz denominado "cells", que contiene un arreglo con tantas entradas como posiciones se hayan definido para postprocesar en la configuración.

Cada una de estas entradas representa una posición concreta dentro de la fibra cardiaca, identificada por su identificador ("position") específica dentro de la fibra simulada. Dentro de cada posición se encuentran almacenados distintos marcadores electrofisiológicos obtenidos del análisis de cada potencial de acción detectado durante la simulación (como se ha mencionado anteriormente). En particular, el archivo contiene las siguientes variables clave:

- **"time"**: Instantes temporales en los que se detecta el inicio de cada potencial de acción analizado.
- **"APD10", "APD25", "APD50", "APD75", "APD90"**: Duraciones del potencial de acción calculadas a distintos niveles de repolarización (10%, 25%, 50%, 75% y 90% respectivamente).
- **"Triangulation"**: Diferencia entre APD90 y APD50, un indicador importante sobre la forma del potencial de acción.
- **"V_max"** y **"V_min"**: Valores máximo y mínimo alcanzados por el potencial transmembrana durante cada potencial de acción.
- **"dVdt_max"** y **"dVdt_min"**: Pendientes máxima y mínima del potencial transmembrana durante cada potencial de acción, útiles para caracterizar la velocidad de subida y bajada del potencial.

Cada uno de estos marcadores se representa mediante un arreglo numérico que contiene los valores obtenidos para cada potencial de acción detectado durante la simulación.

5.2.5. Integración de la aplicación con BOINC

Para que el simulador funcione correctamente dentro del entorno proporcionado por la infraestructura de BOINC, se han realizado una serie de adaptaciones esenciales en el código. Estas modificaciones permiten que el programa pueda comunicarse eficazmente con el servidor de BOINC, interactuando con la plataforma para recibir tareas, gestionar archivos, informar del progreso, y guardar puntos de control (*checkpoints*) para preservar el estado de la simulación.

Adaptación del main de la aplicación

La primera adaptación fundamental ha consistido en modificar el archivo principal (*main*) del simulador siguiendo la plantilla proporcionada por la aplicación de ejemplo "BOINC Hello World!". Este ejemplo ha sido clave para entender cómo realizar correctamente la comunicación del simulador con el servidor central del proyecto DENIS@Home.

En primer lugar, se crea una instancia de la clase *BoincConnection* (de la cual hablaremos posteriormente), la cual gestiona todas las interacciones con la API y la librería de BOINC. Posteriormente, esta instancia proporciona el acceso al archivo de entrada (*in*) que contiene la configuración específica de la simulación. A partir de esta configuración se inicializa una instancia de la clase *Simulation*, encargada de ejecutar la simulación (ver apartado 5.2.2). Durante la ejecución del programa, se verifica si existe un checkpoint anterior disponible. En caso afirmativo, se recupera dicho estado y la simulación continúa desde el punto donde se interrumpió. Si no existe un checkpoint válido, se inicia una nueva simulación desde cero (explicado más adelante en detalle).

Al finalizar la simulación, se volcarán los datos correspondientes al output y al postprocesado para que posteriormente sean mandados al servidor, y ya se validen y asimilen.

Integración con la API de BOINC mediante la clase BoincConnection

La comunicación entre el simulador y la API proporcionada por BOINC se lleva a cabo mediante la clase *BoincConnection*.

Al inicializarse, *BoincConnection* lleva a cabo dos acciones principales. Primero, ejecuta el método *initAppData()*, responsable de inicializar las funciones de diagnóstico estándar de BOINC y obtener los datos básicos de inicialización de la aplicación, como rutas de trabajo, tiempos iniciales, y configuración específica de la tarea recibida desde el servidor. En caso de que esta inicialización falle, la clase interrumpe inmediatamente la ejecución, garantizando que la simulación no continúe en condiciones incorrectas.

La segunda acción fundamental es la ejecución del método *resolveFileNames()*, encargado de resolver las rutas absolutas de los archivos de entrada (*INPUT_FILENAME*), salida (*OUTPUT_FILENAME*), postprocesado (*MARKERS_FILENAME*), y checkpoints (*CHECKPOINT_FILE* y *CHECKPOINT_FILE_2*) dentro del entorno BOINC. De este modo, el simulador podrá acceder a cada archivo necesario para su ejecución desde cualquier ordenador voluntario.

Durante la simulación, *BoincConnection* ofrece métodos específicos para acceder a cada archivo (*getInputFile(), getOutputFile(), getMarkersFile()*), gestionar checkpoints (*getCheckpointFile(), timeToCheckpoint(), markCheckpointCompleted()*), informar del progreso al servidor de BOINC

(*updateFractionDone()*), y proteger las secciones críticas mediante métodos explícitos (*startCriticalSection(), endCriticalSection()*).

Guardar el estado de la simulación (Checkpoints)

En entornos de cómputo distribuido, cabe la posibilidad de que una simulación de larga duración se vea interrumpida accidentalmente (por falla del hardware, corte de energía o límite de tiempo del host voluntario). Para evitar la pérdida total de todo el trabajo ya realizado, BOINC ofrece la posibilidad de implementación de un mecanismo de **checkpoints**: puntos de control intermedios que almacenan el estado completo de la simulación en disco. Si la tarea se detiene, el cliente puede reanudarla más tarde desde el último checkpoint guardado, en lugar de comenzar desde cero. Este mecanismo ha de ser implementado, ya que BOINC solo te ofrece saber cuándo es el momento en el que se debe de guardar, dejando así al desarrollador la posibilidad de implementación de guardado y carga de los mismos.

En nuestra aplicación, este mecanismo se integra directamente en la clase *Simulation*. Durante cada iteración del bucle principal, justo después de actualizar todos los estados de las células y registrar el progreso con BOINC, se invoca *boincConnection->timeToCheckpoint()*. Este método pregunta a BOINC si es el momento apropiado para realizar un checkpoint (en función del tiempo de CPU consumido y de la frecuencia configurada por el voluntario para ese cliente). Si BOINC confirma que se debe guardar un punto de control, la simulación llama internamente al método *saveCheckpoint()* de la propia clase.

El método *saveCheckpoint()* del simulador realiza dos tareas consecutivas:

- Comienza una sección crítica con *boincConnection->startCriticalSection()*, asegurando que ningún otro hilo o proceso intente escribir simultáneamente en los mismos archivos.
- Abre dos archivos de checkpoint: uno principal (*denis_checkpoint*) y un secundario (*denis_checkpoint_2*). El uso de dos archivos es una práctica recomendada para minimizar el riesgo de corrupción: si la escritura en el primero falla, aún se conserva el segundo.

A continuación, el simulador serializa y escribe el estado completo en el archivo de checkpoint:

- Primero, graba el número de iteración (*iteration*), que indica cuántos pasos temporales ha ejecutado hasta el momento.
- Segundo, para cada posición en la fibra, escribe el vector de estados (*states[i]*) de longitud igual al número de variables de estado del modelo. De esta forma, cuando se reanude la simulación, se conocen todos los valores actuales de cada variable de cada célula.

- Finalmente, delega a *output->saveCheckpoint()* y *postprocessing->saveCheckpoint()* la escritura de los datos parciales de salida y postprocesado respectivamente, en el mismo flujo de bytes. El método *Output::saveCheckpoint()* escribe todos los valores intermedios que ya estaban almacenados en memoria (*states*, *rates* y *algebraics*, junto con el vector de tiempos). De manera similar, *Postprocessing::saveCheckpoint()* guarda los marcadores electrofisiológicos calculados hasta ese punto.

Tras escribir todos los bloques de datos, se cierra el archivo principal y se marca el checkpoint como completado mediante *boincConnection->markCheckpointCompleted()*. Solo después de esto BOINC podrá marcar la tarea como "guardada" y, si se interrumpe, reiniciarla más adelante. Finalmente, se cierra el archivo secundario de la misma manera.

Cuando la aplicación se inicia (o se reinicia) en un ordenador voluntario, el propio main de la aplicación llama a *simulation->loadCheckpoint()*. Este método comprueba primero si la unidad de trabajo ya lleva tiempo de CPU consumido (*boincConnection->getWorkUnitCPUTime() > 0*); si no es así (caso de nueva ejecución), devuelve true y no hace nada. Si existe tiempo de CPU previo, intenta abrir el archivo de checkpoint principal (*denis_checkpoint*). Si puede leerlo, deserializa en orden:

- El número de iteración, reestableciendo *iteration*.
- Los vectores *states[i]* para cada célula, de modo que sus variables de estado vuelvan a los valores exactos de la última escritura.
- Invoca *output->loadCheckpoint()* y *postprocessing->loadCheckpoint()* para recargar cualquier dato intermedio de salida y postprocesado, de forma que la memoria interna quede íntegra.

Si la lectura del archivo principal falla, intenta lo mismo con el secundario (*denis_checkpoint_2*). Solo cuando alguno de los dos checkpoint es cargado con éxito, el método devuelve true y la simulación continúa desde el punto donde se detuvo. En caso de que ninguno de los dos checkpoint sea legible, se informa de *"Failed to load checkpoint"* y la simulación vuelve a crearse desde cero.

Gracias a esto, se garantiza que el trabajo realizado en ordenadores voluntarios no se pierde en caso de interrupciones imprevistas.

5.2.6. Dependencias de la aplicación

Para que el simulador funcione correctamente, ha sido necesario integrar y compilar dos librerías externas fundamentales: jsoncpp y BOINC (incluyendo también su API). La librería jsoncpp permite manejar y generar la salida del archivo de postprocesado en formato JSON, mientras que la librería BOINC es esencial para establecer toda la comunicación requerida con la infraestructura

distribuida, permitiendo la ejecución del simulador en los ordenadores voluntarios conectados al proyecto.

Con el objetivo de facilitar la compilación del simulador, se ha creado una estructura de carpetas específica, organizada para cada dependencia:

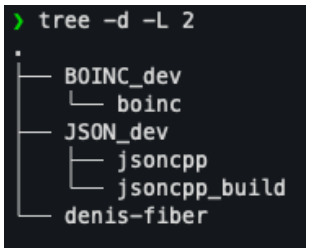


Figura 15: Comando usado para mostrar la estructura de carpetas requerida por el simulador

La primera carpeta, *BOINC_dev*, contiene el código fuente y las librerías compiladas del proyecto BOINC. Este proyecto, al requerir configuraciones específicas según la plataforma objetivo (Linux, Windows o macOS), necesita ser compilado previamente para el entorno objetivo, generando así las librerías específicas para que puedan ser enlazadas posteriormente durante la compilación del simulador.

La segunda carpeta, *JSON_dev*, está dedicada a la librería "jsoncpp" [38]. Dentro de ella encontramos los dos directorios siguientes: el primero (*jsoncpp*) almacena el código fuente original, clonado directamente desde el repositorio oficial de GitHub; el segundo directorio (*jsoncpp_build*) contiene los archivos generados tras compilar la librería jsoncpp mediante CMake. Se decidió separar el directorio donde se encuentra el código fuente del que esta el proyecto compilado debido a errores de compilación, y es que cuando se trataba de compilar jsoncpp creando una capeta en el interior de la misma en la que se encuentra el código fuente (es decir, la de *jsoncpp*) CMake lanzaba un error, que indicaba que esto no se podía realizar. Es por eso, que la manera más sencilla de abordar la compilación fue separándola a una carpeta que estuviera fuera (estructura actual de carpetas en el interior de JSON_dev de la Figura 15).

Una vez preparada esta estructura, se utiliza CMake para compilar el simulador, siguiendo las instrucciones definidas en el archivo CMakeLists.txt. Este archivo define las dependencias internas necesarias para generar el ejecutable del simulador. Entre estas instrucciones, destacan especialmente:

- **Definición del estándar del lenguaje**: El proyecto usa C++11 como estándar mínimo para su código fuente.
- **Directorios de salida**: Se especifican rutas para las carpetas de salida que almacenarán librerías y binarios generados tras la compilación.

- **Inclusión de directorios y librerías**: Se añaden explícitamente rutas relativas que apuntan a los directorios donde se encuentran tanto los archivos de cabecera (.h) como las librerías previamente compiladas (BOINC y jsoncpp).
- **Compilación y enlazado del ejecutable**: El archivo define qué librerías internas y externas (denis_fiber_lib, models_lib, boinc_api, boinc, jsoncpp) deben ser enlazadas finalmente para construir el ejecutable.

5.3. Integración en el proyecto de BOINC: DENIS@Home

5.3.1. Adaptaciones del servidor

Para incorporar el nuevo simulador DENIS Fiber al entorno productivo del proyecto BOINC existente (DENIS@Home), fue necesario desplegar una nueva aplicación en el servidor, garantizando al mismo tiempo la continuidad y estabilidad de las tres aplicaciones previas del proyecto:

- *Beta of DENIS-myocyte* (versión anterior del simulador en 0D)
- *New human ventricular cell model*
- *Human ventricular cell models optimization*

Esta tarea presentaba ciertos riesgos operativos, ya que cualquier error podría afectar negativamente al resto de aplicaciones activas o incluso causar interrupciones en el servicio a los voluntarios. Por esta razón, antes del despliegue en producción, se llevó a cabo una validación del procedimiento en el entorno de pruebas específico desarrollado para este fin, y que se detalla en profundidad más adelante en el apartado 5.4. Una vez asegurado que el proceso funcionaba correctamente en el entorno controlado, se procedió al despliegue definitivo. Por ello, los pasos siguientes corresponden a los que se realizaron en primera instancia en el entorno de pruebas, los cuales únicamente difieren de los reales en el nombre que se le puso al proyecto (denisathome) y a la aplicación (DENIS_Fiber_Beta).

El procedimiento de despliegue en el servidor siguió los estándares recomendados por la documentación oficial de BOINC (Create a BOINC Server Cookbook) [39], mediante los siguientes pasos. En primer lugar, se añadieron los archivos de plantilla (templates) para la nueva aplicación en la carpeta correspondiente del proyecto, denominándolos según la convención: denis-fiber_in y denis-fiber_out. Estos templates especifican la estructura de entrada y salida esperada para la comunicación de las tareas con los voluntarios.

Posteriormente, a través de la interfaz administrativa del servidor BOINC, se creó la nueva aplicación, definiendo su nombre como denis-fiber (en el caso de la original sería "DENIS_Fiber_Beta") y una descripción clara para identificarla fácilmente entre las demás aplicaciones existentes en la plataforma.

A continuación, se integraron dos componentes necesarios para el correcto funcionamiento de cualquier aplicación BOINC: el validador y el asimilador (los cuales se detallan más adelante en los apartados 5.3.3 y 5.3.4 respectivamente). El primero, específico para DENIS Fiber y desarrollado previamente, se compiló por separado (mediante el comando "make") y se trasladó el binario resultante (denis-fiber-validator) al directorio bin/ del proyecto. Para el asimilador se optó por usar uno básico ya existente en el servidor (sample_assimilator), el cual únicamente se copió con en nombre de "denis-fiber-assimilator", para respectar la nomenclatura de la aplicación. Para activar estos componentes se añadió la siguiente configuración en el archivo principal del servidor (config.xml):

```
<daemon>
    <cmd>denis-fiber-validator --app denis-fiber</cmd>
</daemon>
<daemon>
    <cmd>denis-fiber-assimilator --app denis-fiber</cmd>
</daemon>
```

Figura 16: Daemons correspondientes al validador y asimilador respectivamente

Además, se creó la carpeta *results_denis_fiber* en la raíz del proyecto, necesaria para que el asimilador pueda depositar los resultados recibidos desde los voluntarios.

Una vez configurado el entorno, se procedió a crear la primera versión para la nueva aplicación, con sus correspondientes plataformas soportadas (detalladas en el apartado 5.5), siguiendo exactamente la estructura de directorios establecida por BOINC. A continuación, se muestra un ejemplo:

```
denis-fiber
└── 0.03
    ├── aarch64-unknown-linux-gnu
    │   ├── denis-fiber_0.03_aarch64-unknown-linux-gnu
    │   └── version.xml
    ├── arm64-apple-darwin
    │   ├── denis-fiber_0.03_arm64-apple-darwin
    │   └── version.xml
    ├── windows_x86_64
    │   ├── denis-fiber_0.03_windows_x86_64.exe
    │   └── version.xml
    ├── x86_64-apple-darwin
    │   ├── denis-fiber_0.03_x86_64-apple-darwin
    │   └── version.xml
    └── x86_64-pc-linux-gnu
        ├── denis-fiber_0.03_x86_64-pc-linux-gnu
        └── version.xml
```

Figura 17: Ejemplo de la estructura de carpetas que debe de seguir una aplicación de BOINC

Esta estructura se debe de crear dentro de la carpeta de aplicaciones (apps/) del proyecto. En esta carpeta de versión, se añadieron también los siguientes dos elementos:

- El archivo version.xml, que describe al servidor los detalles del ejecutable (plataforma, versión, nombre del archivo y posibles atributos de prueba, como versiones beta).
- El ejecutable binario compilado previamente, con el nombre exactamente coincidente al especificado dentro del archivo version.xml.

Con los archivos en posición, se llevó a cabo la firma criptográfica obligatoria del binario siguiendo las directrices oficiales (disponibles en la documentación oficial de BOINC) [40], puesto que, sin esta firma, muchos usuarios no ejecutarían en sus dispositivos el programa que les ha llegado. El archivo se ha de llamar igual que el ejecutable binario, pero terminado en ".sig", además de depositarse en la misma carpeta que el mismo.

Una vez realizada la firma, se actualizó la base de datos del servidor ejecutando el script proporcionado por BOINC desde la raíz del proyecto. Para ello, se ejecutaron los comandos *"$ cd ~/projects/test"* y *"$ bin/update_versions"* en ese orden.

Por último, para que los cambios aplicaran dentro del proyecto, fue necesario reiniciarlo. Para ello, se ejecutaron los siguientes comandos *"$ bin/stop"* y *"$ bin/start"* en ese orden.

Con estos pasos finalizados, la aplicación quedó completamente integrada y operativa en el servidor BOINC de DENIS@Home, preparada para recibir tareas y ser utilizada por los voluntarios.

5.3.2. Mandar tareas a los voluntarios

El proceso de generación de tareas para los voluntarios comienza creando un fichero de entrada XML que describe la simulación concreta que queremos ejecutar (descrito en el apartado 5.2.1). Este XML, por ejemplo, debemos nombrarlo como *infile* y colocarlo en la carpeta de trabajo del servidor. En este paso, el investigador define todos los parámetros de la fibra cardiaca necesarios según sus necesidades.

A continuación, ese fichero se registra (*stage*) en la infraestructura BOINC mediante el comando "*$bin/stage_file infile*". Internamente, este comando mueve *infile* a la ubicación que BOINC utiliza como origen de datos para el proyecto (generalmente la carpeta *download* dentro del proyecto), dejándolo listo para ser referenciado en la creación de los paquetes de trabajo (*workunits*).

Una vez el archivo está registrado, se invoca el binario create_work para generar las workunits que se distribuirán entre los clientes. Por ejemplo:

```
boincadm@boinc-server:~/projects/test$ bin/stage_file infile
boincadm@boinc-server:~/projects/test$ bin/create_work --appname denis-fiber infile
WARNING: MYSQL_OPT_RECONNECT is deprecated and will be removed in a future version.
workunit name: denis-fiber_623_1748510683.820626
boincadm@boinc-server:~/projects/test$
```

Figura 18: Comandos para registrar y crear un nuevo trabajo en el servidor de BOINC

Este comando crea una workunit por cada entrada válida en *infile*. Si se dispone de múltiples archivos, se puede automatizar su envío usando la opción --stdin: "*$bin/create_work --appname denis-fiber --stdin < infile_list*" (las cuales se tendrán que haber registrado previamente antes crear el trabajo, además de guardados sus nombres en una lista de archivos).

En ambos casos, BOINC responde con los identificadores de las workunits creadas (como se puede ver en la Figura 18), confirmando que han pasado a la cola de distribución.

Finalmente, para verificar que las tareas han sido correctamente enviadas a los voluntarios y podemos recibirlas en nuestro equipo, basta con abrir BOINC Manager, seleccionar el proyecto (si no lo tienes simplemente puedes añadirlo) y pulsar "Update" en el menú de comandos del proyecto. Las nuevas workunits aparecerán en la lista de envío y, una vez procesadas por los clientes, sus resultados se recopilarán automáticamente en la carpeta sample_results/ del servidor.

Toda tarea es enviada en DENIS@Home a 2 equipos voluntarios (es un parámetro de configuración) para poder comprobar que los resultados recibidos son correctos. Esta comprobación se realiza mediante el validador de resultados que se describe en el apartado siguiente.

```
boincadm@boinc-server:~/projects/test$ ls -la sample_results/
total 1164
drwxrwxr-x  2 boincadm boincadm    4096 May 29 09:36 .
drwxrwxr-x 15 boincadm boincadm    4096 May 29 09:23 ..
-rw-rw-r--  1 boincadm boincadm 1179496 May 29 09:36 denis-fiber_623_1748510683.820626_0
-rw-rw-r--  1 boincadm boincadm    2831 May 29 09:36 denis-fiber_623_1748510683.820626_1
boincadm@boinc-server:~/projects/test$ head -10 sample_results/denis-fiber_623_1748510683.820626_0
"time in component environment (millisecond)"   "V in component membrane (millivolt) at position 125"   "V in component membrane (millivolt) at po
sition 150"   "V in component membrane (millivolt) at position 175"   "V in component membrane (millivolt) at position 200" "V in component membra
ne (millivolt) at position 225"
0.000000000000e+00      -8.523000000000e+01     -8.523000000000e+01     -8.523000000000e+01     -8.523000000000e+01     -8.523000000000e+01
1.000000000000e-01      -8.523913968147e+01     -8.523913968147e+01     -8.523913968147e+01     -8.523913968147e+01     -8.523913968147e+01
2.000000000000e-01      -8.524811797859e+01     -8.524811797859e+01     -8.524811797859e+01     -8.524811797859e+01     -8.524811797859e+01
3.000000000000e-01      -8.525693826023e+01     -8.525693826023e+01     -8.525693826023e+01     -8.525693826023e+01     -8.525693826023e+01
4.000000000000e-01      -8.526560364724e+01     -8.526560364724e+01     -8.526560364724e+01     -8.526560364724e+01     -8.526560364724e+01
5.000000000000e-01      -8.527411706400e+01     -8.527411706400e+01     -8.527411706400e+01     -8.527411706400e+01     -8.527411706400e+01
6.000000000000e-01      -8.528248127705e+01     -8.528248127705e+01     -8.528248127705e+01     -8.528248127705e+01     -8.528248127705e+01
7.000000000000e-01      -8.529069892495e+01     -8.529069892495e+01     -8.529069892495e+01     -8.529069892495e+01     -8.529069892495e+01
8.000000000000e-01      -8.529877254145e+01     -8.529877254145e+01     -8.529877254145e+01     -8.529877254145e+01     -8.529877254145e+01
boincadm@boinc-server:~/projects/test$
```

Figura 19: Carpeta sample_results con los resultados correspondientes de la prueba de mandar tareas

5.3.3. Validador de resultados

El validador implementado específicamente para la aplicación se encarga de comprobar la coherencia de los resultados que devuelven los voluntarios. Para ello, por defecto cada unidad de trabajo se envía a dos equipos simultáneos; tras la comparación de ambos resultados, si no se consideran iguales, se manda a un tercero y se vuelve a comprobar, y así hasta agotar el número máximo de reintentos establecidos en la configuración al mandar el trabajo (3 reintentos por defecto) [41]. Puede haber múltiples razones por las que una tarea no sea válida, es un mecanismo de control para asegurar que los voluntarios no intenten hacer trampas, pero muchas veces no solo depende del equipo del usuario o su intención, también permiten detectar problemas en el desarrollo de la aplicación.

Para que este funcione, necesitamos tener la librería de jsoncpp construida. En este caso, como el servidor se encuentra funcionando sobre un Ubuntu, podemos directamente instalarla en el sistema mediante el comando "*$sudo apt install libjsoncpp-dev*". De esta manera, gracias a las facilidades de Linux, nos evitamos tener que construir la estructura como hemos descrito en el apartado 5.2.6 para la librería de jsoncpp.

A la hora de validar, este analiza y compara dos tipos de archivos generados durante la simulación: los archivos de salida tabular (output) y los archivos JSON generados en el postprocesado, los cuales han sido explicados con más detalle en las secciones 5.2.3 y 5.2.4 respectivamente.

En primer lugar, el validador identifica automáticamente el tipo de cada archivo mediante la lectura de su primer carácter. Posteriormente, procesa el contenido del archivo según su tipo, almacenándolo en estructuras internas. Finalmente, compara numéricamente estas estructuras con resultados obtenidos por otros voluntarios mediante una comparación valor a valor. Esta comparación admite pequeñas desviaciones (tolerancia relativa del 0.1%) para manejar diferencias menores debidas a errores de redondeo propios del cálculo numérico en diferentes plataformas.

Si los resultados comparados coinciden dentro de dicha tolerancia, el validador marca la tarea como correcta y válida. En caso contrario, la tarea es marcada como inválida, y el servidor decide si reenviarla o descartarla definitivamente (según lo comentado anteriormente). De este modo se garantiza que únicamente resultados fiables se incorporen a la base de datos final del proyecto.

5.3.4. Asimilador de archivos

El asimilador utilizado para DENIS Fiber es el que proporciona por defecto por la plataforma BOINC. Este asimilador básico realiza principalmente dos tareas fundamentales:

- Si la simulación finaliza correctamente, almacena los archivos generados por la ejecución del voluntario en una carpeta específica llamada results_denis_fiber. Estos ficheros los guarda con el mismo nombre de la unidad de trabajo (WU_NAME) seguidos de la extensión según el tipo de fichero (out o markers) definidos en los templates del proyecto.
- Si la simulación falla o excede el número permitido de errores, registra el incidente añadiendo una línea en el archivo results_denis_fiber/errors, incluyendo el nombre de la tarea y el código del error obtenido.

5.4. Entorno de pruebas con Docker Compose

Para validar cada nueva versión de la plataforma sin comprometer la infraestructura real de DENIS@Home se ha construido un entorno de pruebas aislado mediante Docker Compose. La solución encapsula los dos servicios fundamentales de cualquier servidor BOINC, que son una base de datos MySQL y un servidor de aplicaciones (generalmente Ubuntu), para posteriormente orquestar su arranque, configuración y persistencia de datos. De este modo es posible reproducir en cuestión de minutos las mismas condiciones de ejecución que existen en el servidor real, ejecutar pruebas, inspeccionar registros o depurar sin riesgo de afectar a la red de voluntarios reales.

Estas imágenes de Docker están basadas en la guía oficial de BOINC comentada anteriormente (*Create a BOINC Server Cookbook*) [39], para construir un servidor desde cero en una maquina Ubuntu. Aunque posteriormente se hayan tenido que añadir más elementos para replicar DENIS@home, la base de estas imágenes corresponde a la misma. Para su adaptación a la plataforma de Docker, se define el fichero compose.yaml, que define estos dos servicios:

- **mysql**, que arranca a partir de una imagen personalizada donde se aplican directrices básicas de endurecimiento (contraseña de root, eliminación de usuarios anónimos) y expone el puerto 3306 para el acceso del servidor BOINC. Su volumen persistente mysql_data garantiza que las tablas sobrevivan a reinicios del contenedor y permiten comparar el estado antes y después de una prueba.
- **boinc-server**, construido desde un Dockerfile que replica paso a paso la guía sobre Ubuntu 22.04. El contenedor recibe mediante argumentos de compilación las credenciales y la dirección del servicio MySQL, instala todo lo necesario, ejecuta los scripts de despliegue automatizado y expone los puertos habituales: 22 (SSH para administración), 80 (panel web y planificador) y 31416 (RPC). Tres volúmenes adicionales (*boinc-server-data, crontab_data, apache_data*) preservan la configuración y los datos del proyecto, tareas cron necesarias y los ajustes de Apache para la interfaz web de la que provee el servicio.

A continuación, se contará más en detalle que es lo que aporta cada uno de estos servicios a la plataforma.

5.4.1. Base de datos: MySQL

La primera pieza del entorno de pruebas es una instancia MySQL que reproduce, de forma no interactiva, las mismas condiciones de endurecimiento exigidas por la guía.

El contenedor se construye a partir de la imagen oficial mysql:latest. En el proceso de construcción del mismo, se establece la contraseña de root mediante la variable de entorno

MYSQL_ROOT_PASSWORD y se añade el script secure_mysql.sh al directorio /docker-entrypoint-initdb.d/. De esta manera lograremos que el script se ejecute automáticamente la primera vez que el contenedor arranca, haciendo así los cambios efectivos cuando se levantan los contenedores.

El contenido de secure_mysql.sh traslada, en la medida de lo posible, a comandos SQL los pasos interactivos del utilitario mysql_secure_installation (detallados en el cookbook) [39]:

1. Refuerzo de la cuenta root: se garantiza que sólo pueda autenticarse desde localhost.
2. Eliminación de usuarios anónimos, evitando accesos no controlados.
3. Creación del usuario de servicio boincadm con la misma contraseña que en la guía y permisos globales (*GRANT ALL PRIVILEGES ON *.**). A diferencia del cookbook, el host se fija para permitir que el contenedor boinc-server se conecte a MySQL a través de la red interna de Docker Compose.
4. Actualización de tablas de privilegios con *FLUSH PRIVILEGES*, de modo que los cambios sean efectivos sin reiniciar.

El volumen mysql_data hace que los datos de la base de datos persistan cuando se para y se relanza, o se borra el contenedor. Asimismo, el puerto 3306 se expone para su acceso desde la máquina Ubuntu.

5.4.2. Ubuntu como servidor del proyecto

Para la segunda pieza de este entorno, se emplea un contenedor Docker basado en la imagen oficial de Ubuntu 22.04 LTS, la distribución recomendada explícitamente por el equipo de desarrollo de BOINC. Este servidor es responsable de albergar toda la lógica operativa del proyecto, incluyendo la interacción con voluntarios, la gestión de tareas y resultados, así como la interfaz web de administración.

La imagen personalizada parte de una configuración básica, estableciendo primero las variables de entorno necesarias para permitir una instalación desatendida. Seguidamente se crea y configura un usuario denominado *boincadm*, con privilegios suficientes para llevar a cabo todas las acciones necesarias dentro del contenedor relativas al proyecto.

Tras la preparación inicial del usuario y el entorno básico, se instalan mediante comandos de sistema (*apt*) todas las dependencias requeridas por BOINC, siguiendo la guía oficial.

Una vez instaladas las dependencias, se procede a la configuración específica del servidor BOINC. El código fuente oficial del servidor se obtiene directamente desde su repositorio oficial de GitHub, manteniendo así coherencia con la versión que se usa actualmente en el entorno real de DENIS@Home (*server_release/1.4/1.4.4*). A partir de aquí, se compila BOINC, excluyendo

componentes innecesarios como el cliente BOINC y la interfaz gráfica, ya que estos no son requeridos dentro del entorno del servidor.

Una vez compilado el servidor de BOINC, se crea una configuración de SSH, que nos permitirá conectarnos posteriormente al entorno desde nuestra maquina o en remoto al servidor. Para ello, se usarán de nuevo las credenciales por defecto detalladas en la guía, para el usuario boincadm.

Posteriormente, se añaden varios scripts adicionales que automatizan el despliegue inicial del proyecto dentro del contenedor:

- El script *while_starting.sh*, es el que está configurado como punto de entrada del contenedor, que se encarga de iniciar el servicio SSH y el servicio Apache2.
- El script *auto_setup.sh*, es el que se encarga de crear el proyecto de BOINC, según se detalla en el cookbook [39]. Este se debe de ejecutar una vez lanzado el contenedor de Docker, ya que durante la creación del proyecto se descubrieron varias limitaciones, y una de ellas era que no dejaba crear el proyecto durante la construcción de la imagen. Este se detalla más adelante en el apartado 5.4.3.
- El script *setup_files.sh*, se encarga de coger los archivos personalizados que se usan en el proyecto y reemplazarlos por los ya existentes, o añadir los nuevos (para hacerlo así lo más realista y similar a la web donde se alberga el proyecto principal).

También, a parte de los scripts para la automatización del despliegue, se añaden varios archivos que también son esenciales para el correcto funcionamiento del proyecto:

- El archivo project.inc, el cual añade contenido personalizado para mostrar en el proyecto (como el nombre de este, copyright…)
- El archivo .htpasswd, el cual contiene al usuario boincadm con la contraseña hasheada "foobar99", que servirá de utilidad para el panel de administración de la web. Esto se hace ya que durante la construcción de la imagen no se puede usar el comando htpasswd (requerido para generar ese hash para el usuario), que requiere la entrada por teclado de la contraseña, por lo que se ha automatizado en la medida de lo posible.
- El archivo sudoers, creado para hacer que el usuario boincadm tenga permisos de usuario privilegiado usando el comando sudo. Este archivo se reemplaza por el original creado en la instalación de sudo, ya que durante la construcción de la imagen no se puede editar texto, por lo que se ha automatizado en la medida de lo posible.
- Se clonan los archivos necesarios para incluir en la web, del repositorio del grupo denis_web. Su finalidad es únicamente hacer que el sitio web se vea lo más similar al real. Estos archivos serán usados por el script mencionado anteriormente *setup_files.sh*.

Por último, la imagen expone explícitamente los puertos 22 (SSH), 80 (HTTP) y 31416 (RPC del servidor BOINC), facilitando así tanto la administración remota del contenedor como la interacción directa con la aplicación BOINC desde el exterior.

5.4.3. Script de inicialización del proyecto

Como bien se ha detallado anteriormente, el script *auto_setup.sh* es el que se encarga de inicializar correctamente el proyecto de BOINC según los pasos del cookbook [39]. Para ello, en primer lugar, este script crea el proyecto ejecutando el comando *make_project*, proporcionándole las credenciales de la base de datos MySQL que fueron configuradas previamente (*boincadm, foobar99*), la dirección del host (*mysql-server*) y la URL base local (*http://127.0.0.1*).

Posteriormente, habilita la página web del proyecto para que sea accesible mediante Apache. Para ello, copia la configuración generada automáticamente (*test.httpd.conf*) al directorio correspondiente del servidor web, reiniciando Apache para aplicar los cambios inmediatamente.

A continuación, se añade automáticamente una tarea periódica en el sistema (*crontab*), que ejecutará cada cinco minutos el script *start --cron* proporcionado por BOINC dentro de los binarios del proyecto, cuyos fines son únicamente de mantenimiento. Luego, el script ejecuta el binario xadd, también proporcionado por BOINC, que se encarga de inicializar la base de datos del proyecto.

Seguidamente, se habilita el panel de administración en la web. Para asegurar el acceso restringido al panel de administración (*/test_ops*), se copia un archivo *.htpasswd* previamente configurado y añadido. Tras ello, también se personaliza el proyecto con el documento mencionado anteriormente *project.inc*.

Finalmente, el script arranca definitivamente la aplicación BOINC mediante el binario start y vuelve a reiniciar Apache para que los servicios web asociados al proyecto estén completamente operativos y accesibles desde el exterior del contenedor a través de la dirección local proporcionada.

5.5. Compilación multiplataforma y automatización

5.5.1. Imágenes de Docker para compilación en entornos Windows y Linux

Para asegurar la máxima reproducibilidad, la generación de binarios para Windows y Linux se lleva a cabo mediante contenedores Docker especialmente preparados para cada entorno.

En el caso de Windows (x86_64), se emplea un contenedor basado en la imagen oficial de Windows Server Core con .NET Framework 4.8, en la cual se instala Chocolatey para gestionar dependencias como Git y CMake. Además, se utilizan las herramientas de compilación de Visual Studio (Build Tools) necesarias para construir aplicaciones nativas en Windows (aunque en

primera instancia se usó MinGW, pero llevo a problemas de rendimiento notables: ver Anexo IV). Dentro del contenedor, primero se compilan las librerías auxiliares como BOINC (en particular, los componentes libboinc y libboincapi) y jsoncpp, debido a que son requisitos para la aplicación del simulador.

Tras la preparación de estas librerías, se genera un script automatizado en PowerShell (compiler.ps1) que realiza la descarga del código fuente de DENIS Fiber, selecciona una versión específica del código (mediante la adaptación de GitFlow comentada en el apartado 4.5), ejecuta la configuración mediante CMake y realiza la compilación final del binario. Finalmente, dicho binario se guarda en una ruta accesible con la nomenclatura específica requerida por BOINC.

En cuanto a las arquitecturas Linux (x86_64 y ARM64), el proceso es conceptualmente similar, aunque adaptado a las particularidades del sistema operativo. Cada arquitectura utiliza una imagen base distinta según sus requisitos específicos:

- Para x86_64-pc-linux-gnu, se emplea una imagen Ubuntu 20.04, desde la cual se instalan todas las dependencias necesarias mediante el gestor de paquetes oficial (apt).
- En cambio, para la arquitectura ARM64 (aarch64-unknown-linux-gnu) se optó por Ubuntu 16.04 (Xenial), ya que es la versión con mayor compatibilidad con plataformas ARM, aunque esto supuso compilar manualmente CMake desde el código fuente para disponer de una versión compatible con la librería jsoncpp.

En ambas arquitecturas de Linux, tras la preparación de las dependencias (BOINC y jsoncpp), se genera dinámicamente un script bash (compiler.sh), muy similar al de Windows, encargado de clonar el repositorio de DENIS Fiber, seleccionar la etiqueta correcta del código, compilar mediante CMake y finalmente guardar el binario en una ubicación con la nomenclatura específica. Tras completar este proceso, los scripts realizan una limpieza del espacio de trabajo eliminando los archivos generados durante la compilación.

5.5.2. Compilación para entornos macOS

En sistemas macOS, la automatización de la compilación presenta particularidades, ya que Apple emplea dos arquitecturas principales: Intel (x86_64) y ARM (Apple Silicon). En este caso, la solución diseñada se basa en scripts escritos en *bash* ejecutados localmente en un sistema macOS con las herramientas necesarias previamente instaladas, debido a que no existe una solución oficial ni sencilla basada en contenedores Docker para macOS.

El proceso para ambas arquitecturas es similar: se parte de un script común denominado apple-darwin.sh, que contiene todos los pasos generales para realizar la compilación. Este script principal realiza la descarga y compilación inicial tanto de las librerías jsoncpp como de las

dependencias de BOINC, para luego, al igual que anteriormente, acabar con la compilación del simulador.

Posteriormente, se emplean dos scripts específicos, x86_64-apple-darwin.sh y arm64-apple-darwin.sh. Dichos scripts establecen una bandera adicional necesaria para especificarlo posteriormente, ya que, en macOS podemos usar el comando *arch*, y es que este nos permite compilar para una arquitectura objetivo de macOS que le especifiquemos (emulando así dicha arquitectura), y finalmente llaman al script principal apple-darwin.sh para completar la compilación.

El resultado final es un binario optimizado para la arquitectura objetivo, almacenado con la nomenclatura específica requerida por el sistema BOINC para su posterior distribución a voluntarios que utilizan sistemas macOS.

5.5.3. Scripts de automatización de la compilación para las diferentes plataformas

Una vez desarrolladas las soluciones de compilación específicas para cada sistema operativo, se abordó el reto de integrar todo este trabajo en una automatización completa, que permitiera obtener, mediante una ejecución sencilla y reproducible, toda la estructura requerida por BOINC para distribuir nuevas versiones del simulador.

BOINC exige una estructura estricta para sus aplicaciones, definida por tres niveles jerárquicos: nombre de la aplicación, versión y arquitectura objetivo. De esta forma, una nueva versión del simulador debe seguir el formato ya mencionado anteriormente (véase la Figura 17).

Dentro de cada carpeta de arquitectura deben encontrarse, al menos, dos elementos: el binario compilado específico y un archivo version.xml requerido por BOINC para identificar correctamente cada ejecutable distribuido. Además de estos dos archivos, es necesario firmar el binario, que es opcional, pero imprescindible. Este paso es el único que se hace dentro del servidor previo a la ejecución del script que te genera la estructura, ya que la clave privada con la que se firma no puede salir del equipo usado para ello [40].

Inicialmente, se planteó centralizar toda la automatización en un único script, pero surgió una dificultad técnica: Docker, en MacOS (que es el entorno que principalmente utilizo), no permite ejecutar contenedores con imágenes de Windows, lo que obligó a dividir la estrategia en dos procesos independientes. La plataforma Windows se compila separadamente en un sistema operativo Windows, mientras que el resto (Linux y macOS) puede realizarse desde un sistema operativo macOS.

Para el entorno Windows, se diseñó un script sencillo que utiliza la imagen Docker previamente creada y ejecuta en su interior un script de PowerShell (compiler.ps1). Este contenedor genera

el ejecutable específico para Windows, que posteriormente se extrae del contenedor hacia la máquina local mediante el comando *docker cp*. Tras completar la extracción, el contenedor se elimina automáticamente.

Para el resto de plataformas (Linux y macOS), la automatización se lleva a cabo desde macOS mediante una estrategia modular basada en varios scripts, que se ejecutan en cascada para generar la estructura completa requerida por BOINC. El script compilationOne.sh es el punto de entrada principal, que recibe como argumentos el nombre del proyecto, la versión específica y la etiqueta de código (FiberTag). Este script se encarga de crear la jerarquía básica de directorios y llama individualmente a los scripts secundarios de compilación para cada plataforma: aarch_unkown_linux_gnu.sh, x86_64-pc-linux-gnu.sh, arm64-apple-darwin.sh y x86_64-apple-darwin.sh

Cada uno de estos scripts secundarios se encarga, en primer lugar, de generar su directorio específico dentro de la estructura de versión y arquitectura. A continuación, utilizan Docker (en el caso de las arquitecturas Linux) o scripts locales (en el caso de macOS) para compilar DENIS Fiber según la plataforma objetivo.

Como resultado final, después de ejecutar el script compilationOne.sh, se genera una estructura organizada, estandarizada y lista para desplegarse en el entorno de producción; lo único que tendremos que hacer seria crear la carpeta correspondiente para Windows dentro de esta estructura, debido a que no se está compilando desde el mismo script. Gracias a esta automatización, el proceso de lanzamiento de nuevas versiones se convierte en una tarea sencilla, reproducible y escalable.

6. Estudio Económico

En este apartado se presenta un análisis pormenorizado de los costes incurridos en la puesta en marcha y desarrollo del simulador para el proyecto de BOINC, desglosados en infraestructura, material personal y recursos humanos. El objetivo es cuantificar tanto la inversión necesaria como el esfuerzo temporal dedicado, de cara a evaluar la viabilidad y eficiencia económica del trabajo realizado.

La fase de desarrollo del proyecto se ha planificado en torno a una duración de 7 meses (de mediados de noviembre a mediados de junio), lo que equivale a aproximadamente 210 días naturales. Durante este tiempo de duración, dos personas estuvieron participando de forma activa en el proyecto: Iván Royo Gutiérrez, como desarrollador del proyecto, y Jesús Carro Fernández, como director/jefe del proyecto. Asumiendo una dedicación media de 3 horas de trabajo al día, el desarrollador habrá invertido un total estimado de 210 días × 3 h/día = **630 horas**.

A este cómputo se añaden las **40 horas** estimadas de dedicación por parte del director, obtenidas del tiempo dedicado a reuniones de seguimiento, correcciones de la memoria y a la gestión del servidor BOINC. Por lo que, si sumamos ambos pesos de cada parte correspondiente, obtenemos que el proyecto ha tenido un total de **670 horas** de dedicación.

6.1. Costes de la infraestructura

6.1.1. Costes del servidor actual del proyecto

Las características del servidor utilizado, así como el desglose de su costo estimado a fecha de adquisición 19 de febrero de 2015, son las siguientes:

Tabla 1: Desglose de costes del servidor que alberga BOINC

Componente	Descripción	Precio estimado (€)
Procesador	Intel Xeon E5-2620 v3	600,00
Memoria RAM	16 GiB DDR4 ECC	200,00
Controladora RAID	HP P440	300,00
Placa base + chasis	Servidor HP Gen8	800,00
Fuente y ventilación	PSU redundante y sistema de airflow	200,00
Disco (279 GiB SAS 15 000 rpm)	Almacenamiento principal	120,00
Otros componentes	Cables, backplane, ventiladores…	151,60

Total adquisición (IVA incl.)	**2 371,60**

En entornos empresariales, la vida útil contable de un servidor on-premise suele fijarse entre 3 y 5 años, tras los cuales su valor se considera totalmente amortizado. Si fijamos la vida útil del servidor en 5 años, teniendo en cuenta que estos están funcionando las 24 horas del día, esto es equivalente a unas 43800 horas de uso totales. Aunque, si tenemos en cuenta esta amortización de 5 años, estando a fecha de junio de 2025, podríamos considerar que dicho servidor está completamente amortizado.

6.1.2. Costes mensuales de un servidor en la nube

Para comparar el gasto de mantener un servidor físico con la alternativa de usar un servicio en la nube, se ha elegido una configuración de AWS que reproduzca prestaciones similares a las del servidor actual. En concreto, se selecciona una instancia EC2 de la familia m5.xlarge (4 vCPU y 16 GiB de RAM), ya que ofrece un rendimiento aproximado al Intel Xeon E5-2620 v3 con 6 núcleos/12 hilos y 16 GiB de memoria.

En primer lugar, el precio de la instancia m5.xlarge en AWS en la región EU (Madrid) es de 0,192 USD por hora. Para convertir este valor a euros, se utiliza el tipo de cambio vigente a junio de 2025, que es de 1 USD = 0,8769 EUR. De este modo, el coste por hora en euros se calcula como: $0{,}192\$/\text{h} \cdot 0{,}8769€/\$ \approx 0{,}1684€/\text{h}$.

A continuación, se estima el número de horas que la instancia estaría en funcionamiento en un mes estándar. Si suponemos un mes de 730 horas (≈24,33 h×30 días), el coste mensual de cómputo para la instancia m5.xlarge es: $0{,}1684€/\text{h} \cdot 730\text{h} = 122{,}85€/\text{mes}$.

Por otro lado, el almacenamiento en AWS se gestiona a través de volúmenes EBS (Elastic Block Store). Para reproducir el disco SAS de 279 GiB del servidor físico, se elige un volumen de tipo General Purpose SSD (gp2) de 280 GB. El precio de EBS gp2 en la región EU (Madrid) es de 0,10 USD/GB-mes, lo que equivale a: $0{,}10\text{USD}/\text{GB}_{\text{mes}} \cdot 0{,}8769€/\$ \approx 0{,}0877€/\text{GB}_{\text{mes}}$.

Multiplicando este coste por los 280 GB necesarios, el coste mensual de almacenamiento resulta: $0{,}0877€/\text{GB}_{\text{mes}} \cdot 280\text{GB} = 24{,}56€/\text{mes}$.

En esta estimación no se incluyen costes asociados a IOPS provisionales, snapshots, direcciones IP elásticas ni transferencias de datos de salida (ya que los primeros 100 GB/mes suelen estar incluidos sin coste adicional), asumiendo un uso básico de EBS con I/O estándar. Tampoco se consideran licencias de sistemas operativos, dado que se optaría por una Amazon Linux 2 sin coste adicional de licencia.

El resultado es que un servidor equivalente en AWS (instancia m5.xlarge + 280 GB gp2) implicaría un gasto operativo de 147,41 € al mes.

Tabla 2: Desglose de costes mensuales del servidor en AWS

Concepto	Detalle	Importe (€)
Instancia EC2 (m5.xlarge)	0,1684 €/h × 730 h/mes	122,85
Almacenamiento EBS (280 GB gp2)	0,0877 €/GB-mes × 280 GB	24,56
Total mensual AWS		**147,41**

Para un periodo de 7 meses, esto se traduciría en aproximadamente **1.031,87 €** en costes cloud.

6.2. Costes de recursos humanos

El coste del trabajo de las personas que han estado desarrollando el proyecto puede calcularse a partir de su salario bruto anual y de las horas efectivas trabajadas durante el proyecto.

De media, un desarrollador junior en España cobra en torno a los 30.000€ anuales; además, teniendo en cuenta que este trabaja a jornada laboral completa (40 horas semanales), el convenio para las empresas que tienen este tipo de trabajadores es de 1800 horas anuales de trabajo [42]. Por lo que, el coste por hora de un desarrollador sería de 16,667€/hora.

Por otro lado, para un jefe de proyecto, la media de su salario anual sería de unos 42.000€ brutos; si tenemos en cuenta que también trabaja a jornada laboral completa y con el mismo convenio, serian un total de 1800 horas anuales de trabajo. Teniendo en cuenta esto, podemos deducir que su coste por hora está en unos 23,333€/hora.

Teniendo en cuenta el peso en horas de cada parte en el proyecto, podemos estimar que cada parte ha tenido un coste de:

- **Coste estimado para el desarrollador**: $16{,}667€/hora \cdot 630\ horas\ =\ 10.500{,}21€$
- **Coste estimado para el jefe de proyecto**: $23{,}333€/hora \cdot 40\ horas\ =\ 933{,}32€$

Ahora, bien, a este importe debemos de añadirle la cotización empresarial a la Seguridad Social (SS), que en este año 2025 incluye [43]:

- Contingencias comunes (23,6%)
- Desempleo (5,5% para contrato indefinido, 6,7% para contrato temporal)
- Formación profesional (0,6%)
- Fondo de Garantía Salarial - FOGASA (0,2%)

Por lo que, teniendo en cuenta todo el conjunto de tasas aplicadas, y asumiendo que ambos trabajadores disponen de un contrato indefinido, el porcentaje total que paga la empresa en concepto de SS es de aproximadamente un 29,9% por cada trabajador.

Si añadimos este desglose a los costes calculados anteriormente, obtenemos lo siguiente:

Tabla 3: Desglose de costes de recursos humanos (SS incluida)

Rol	Horas	Tarifa (€/h)	Salario bruto (€)	SS empresa (29,9%) (€)	Coste total (€)
Desarrollador	630	16,667	10.500,21	3.139,5	**13.639,71**
Jefe de proyecto	40	23,333	933,32	279,06	**1.212,38**
Total	**670**		**11.433,53**	**3.418,56**	**14.852,09**

6.3. Costes del material personal

Para el desarrollo correcto del proyecto, se ha requerido tanto un equipo personal de altas prestaciones, como una licencia específica de IDE. A continuación, se desglosan los costes y el criterio de imputación al proyecto de cada uno de estos elementos.

6.3.1. Ordenador Personal

El equipo utilizado para la implementación y prueba del código fue un MacBook Air M2 con 8 GB de RAM y 256 GB de almacenamiento SSD M.2, adquirido en octubre de 2022. A fin de imputar correctamente el coste de este ordenador al proyecto, se ha aplicado un método de amortización lineal acorde con las buenas prácticas en material informático de uso profesional.

En primer lugar, el coste de adquisición del MacBook Air M2 fue de unos 1.300 €. En el sector tecnológico es habitual considerar una vida útil de 4 años para portátiles, debido a la evolución del hardware y al desgaste de sus componentes (batería, velocidad de procesador, etc.). Si tenemos esto en cuenta, el portátil se amortiza en 48 meses a razón de:

$$Cuota\ mensual = \frac{1.300€}{48\ meses} = 27{,}08€/mes$$

Hasta junio de 2025 han transcurrido 32 meses desde la fecha de compra (octubre 2022 – junio 2025), lo que ha supuesto una amortización acumulada aproximada de:

$$27{,}08€/mes \cdot 32\ meses = 866{,}67€$$

En ese momento, el valor neto contable remanente del equipo sería aproximadamente 433,33 € (1 300 € – 866,67 €). Sin embargo, para calcular la parte proporcional que corresponde al periodo

de desarrollo del proyecto (mediados de noviembre 2024 – mediados de junio 2025), se toma la cuota mensual y se multiplica por los 7 meses de uso efectivo en el TFG, por lo que al final nos quedaría:

$$27{,}08€/mes \cdot 7\ meses\ = 189{,}58€$$

De este modo, aunque el ordenador aún no está completamente amortizado, se imputan al proyecto los 189,58 € correspondientes a esos siete meses de dedicación exclusiva a la creación del simulador.

6.3.2. Licencia de IDE para el desarrollo

Para la edición, compilación y depuración del código en C/C++ sobre macOS, se adquirió una licencia anual individual de CLion (entorno de desarrollo integrado de JetBrains). Esta licencia resultó imprescindible para aprovechar funcionalidades avanzadas, autocompletado y depuración nativa, pero sobre todo lo más importante, es porque este IDE trabaja con CMake, misma herramienta que se usa para compilar el proyecto, por lo que esto iba a ser de gran ayuda.

El coste de la licencia fue de 99 € anuales, número de referencia de JetBrains para desarrolladores independientes en la fecha de compra (noviembre 2024).

Además, dado que el periodo de realización del TFG coincide con los primeros 7 meses de vigencia de dicha licencia, se imputará el importe íntegro de 99 € al proyecto. Se considera que sin esta herramienta, el proceso de implementación del simulador habría resultado menos eficiente y más tedioso.

Aunque, para proyectos de investigación o de código abierto, a dichas licencias se les aplica un descuento del 100%; en este caso sí que aplicaría, porque todos los proyectos de BOINC han de ser de código abierto, por lo que la licencia sería gratuita.

6.3.3. Resumen de costes

Por lo que, al final, si juntamos todos los costes, obtenemos que:

Tabla 4: Resumen de costes de material personal

Concepto	Detalle	Importe (€)
Amortización MacBook Air M2 (7 meses)	(1 300 € ÷ 48 meses) × 7 meses	189,58
Licencia CLion (anual)	Adquirida en noviembre 2024 (99 € anuales)	0,00 (Descuento del 100%)
Total costes material personal		**189,58**

El concepto "Amortización del MacBook Air M2 (189,58 €)" representa la parte proporcional del coste de adquisición correspondiente a los siete meses de uso en el desarrollo del TFG, con vida útil de 4 años; mientras que "Licencia de CLion (0 €)" representa coste íntegro de la suscripción anual, con una bonificación del 100% debido a que es un proyecto de código abierto, además de necesaria para garantizar un flujo de trabajo profesional en la programación y depuración de la plataforma BOINC.

6.4. Costes totales del proyecto

Para determinar el coste total del TFG, se suman las tres partidas que hemos desglosado en los apartados anteriores:

- **Costes de la infraestructura** (servidor BOINC o AWS).
- **Costes de recursos humanos** (desarrollador y jefe de proyecto).
- **Costes del material personal** (equipo y licencia de IDE).

Al final, teniendo en cuenta todo esto, y desglosando los costes en función de si tomamos la solución On-Premise o en la nube de AWS, podríamos deducir los siguientes costes:

Tabla 5: Desglose de costes totales para el proyecto

Partida	Importe (€) On-Premise	Importe (€) AWS
Costes infraestructura	0,00	1.031,87
Costes recursos humanos	14.852,09	14.852,09
Costes material personal	189,58	189,58
TOTAL PROYECTO	**15.041,67**	**16.073,54**

Por tanto, la inversión total necesaria para el desarrollo del proyecto asciende a **15.041,67 €** si tomamos la solución del servidor On-Premise o **16.073,54 €** si tomamos la solución del servidor en la nube AWS.

7. Resultados y Evaluación

Al finalizar el desarrollo del proyecto, se ha construido un simulador 1D completamente funcional, e integrado en la plataforma de computación voluntaria ya existente, capaz de ejecutar fibras heterogéneas, teniendo en cuenta que se pueden modificar parámetros de la fibra globalmente (interpaciente) o entre las diferentes células de esta (intrapaciente). Este simulador es capaz de realizar el postprocesado de los resultados en los ordenadores voluntarios, devolviendo así marcadores relevantes en un archivo de formato JSON al servidor. Además, el código se ha verificado y compilado automáticamente para Windows, Linux, Raspberrypi (Linux ARM) y macOS para su posterior distribución dentro de la plataforma. De esta manera, se han cumplido con los objetivos establecidos, además de haber generado un entorno de pruebas que es una característica adicional añadida, y que se encarga de replicar la plataforma original desplegada en el servidor en un equipo local para realizar así pruebas.

Con la plataforma ya operativa y las funciones básicas verificadas, el siguiente paso consiste en evaluar tres casos de uso concretos, de alcance acotado y directamente relevantes para el TFG:

- **Validación frente a ELVIRA**: Para asegurarnos de que nuestro simulador 1D devuelve resultados coherentes, realizaremos una simulación de una fibra sin variaciones y compararemos los resultados con los obtenidos en ELVIRA bajo la misma configuración.
- **Evaluación de checkpoints en simulaciones de larga duración**: Se lanzarán varias simulaciones de 100.000 ms (100 s) sobre una fibra de 3 cm (300 células, dx = 0,01 cm) para verificar el correcto funcionamiento de los checkpoints. Dado que este tipo de simulaciones es demasiado prolongado, el análisis permitirá certificar que, tras cada checkpoint, el guardado y la carga de la información de la simulación se produce sin pérdida de información, ni inconsistencias numéricas.
- **Test de variabilidad de conductancia**: Se lanzarán 10.000 simulaciones de una fibra de 1 cm en las que la conductancia de la corriente Ks (G_{Ks}) se selecciona aleatoriamente en cada posición dentro de un rango de ± 25 %. En este conjunto mediremos:

En todas las pruebas se empleará el modelo TP06 y el método de resolución Forward Euler (FE). Salvo en la prueba de los checkpoints, la fibra tendrá una longitud total de 1 cm, discretizada en 101 posiciones mediante un espaciado $\Delta x = 0{,}01\ cm$, y se establecerá un paso de tiempo $dt = 0{,}001\ ms$. Para reproducir la conducción eléctrica a lo largo de la fibra, se utilizará un coeficiente de difusión de $\sigma = 0{,}0012102\ cm^2/ms$.

El protocolo de estimulación consiste en aplicar un pulso únicamente a la primera célula de la fibra, haciendo así que el estímulo se propague hasta el extremo opuesto. Durante cada simulación, se guardará en el output el último pulso generado en la posición central y se

postprocesarán los cinco últimos pulsos en las posiciones celulares 25, 50 y 75 (habiendo entre cada pulso un segundo de separación).

7.1. Validación frente a ELVIRA

Para asegurar la exactitud de nuestro simulador 1D, se ha diseñado una prueba de comparación directa con ELVIRA. El propósito de esta prueba es verificar que, bajo idénticas condiciones, ambas plataformas generen resultados equivalentes en cuanto a la forma del potencial de acción y los marcadores de repolarización.

En primer lugar, se definirá una configuración para una fibra, sin cambiar nada, además de usar las características mencionadas anteriormente. El estímulo de corriente se aplicará únicamente en la primera célula, con la misma amplitud (I_Stim_Amplitude = -312.0) y la simulación tendrá una duración de 10 s en ambos simuladores. El no cambiar parámetros de la fibra garantiza que cualquier diferencia observada sea atribuible únicamente a las particularidades de implementación numérica de cada plataforma, ya que estas utilizarán diferentes métodos de resolución numérica.

A continuación se muestra la superposición de los potenciales de acción registrados en la posición central (índice 50) de la fibra, obtenidos con ELVIRA (línea continua azul) y con DENIS (línea discontinua naranja).

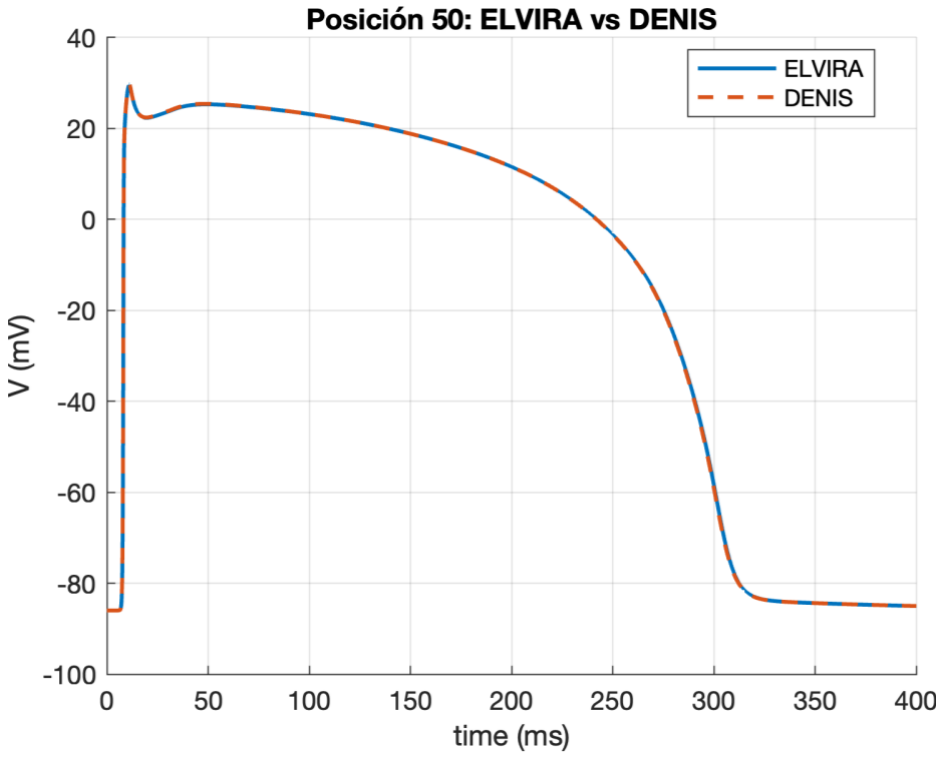


Figura 20: Comparación del potencial de acción en la posición 50 a partir de los resultados obtenidos en ELVIRA y DENIS

Como se puede apreciar, las dos curvas coinciden. Además, para facilitar esta comparación, la figura se ha trazado en un intervalo de 0–400 ms donde se observa que las dos líneas quedan solapadas.

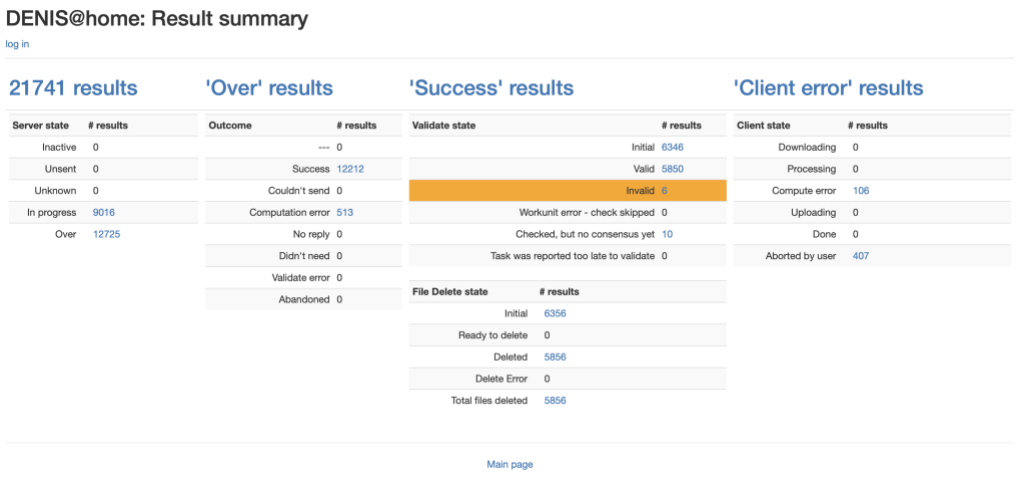
Comparando ambas soluciones numéricamente se pueden ver errores muy bajos, los cuales son normales ya que ambos simuladores utilizando técnicas diferentes. En el caso de ELVIRA se utiliza el método de los elementos finitos con un esquema explícito-implícito, y aplicando la integración de Rush-Larsen [45].

Respecto a los marcadores más relevantes, como el APD90 y el APD50, obtenemos para DENIS un APD90 de 299.04 ms y un APD50 de 274.06 ms, y con en ELVIRA, estaríamos obteniendo un APD90 de 299.4 ms y un APD50 de 274.4 ms. Esta diferencia corresponde a un error prácticamente menor al 0.1%, lo cual podría deberse, de nuevo, al método de resolución numérica.

7.2. Evaluación de checkpoints en simulaciones de larga duración

Para verificar el comportamiento de los puntos de control (checkpoints) en simulaciones prolongadas, se envió un lote de tareas correspondientes a fibras de 3 cm durante 100.000 ms de actividad, que tarda de media unas 12 horas en resolverse. A continuación, se presentan los resultados generales y su interpretación.

En primer lugar, el panel de administración de cualquier aplicación de BOINC te permite mostrar un resumen, para cada aplicación, de los últimos resultados en las últimas veinticuatro horas o los últimos siete días, con información sobre el número de tareas en progreso y finalizadas, así como los resultados de validación y errores de cliente. En este caso, vamos a analizar los resultados para nuestro simulador de los últimos 3 días, ya que la base de datos purga los resultados anteriores (Figura 21).


DENIS@home: Result summary

log in

21741 results

Server state	# results
Inactive	0
Unsent	0
Unknown	0
In progress	9016
Over	12725

'Over' results

Outcome	# results
---	0
Success	12212
Couldn't send	0
Computation error	513
No reply	0
Didn't need	0
Validate error	0
Abandoned	0

'Success' results

Validate state	# results
Initial	6346
Valid	5850
Invalid	6
Workunit error - check skipped	0
Checked, but no consensus yet	10
Task was reported too late to validate	0

File Delete state	# results
Initial	6356
Ready to delete	0
Deleted	5856
Delete Error	0
Total files deleted	5856

'Client error' results

Client state	# results
Downloading	0
Processing	0
Compute error	106
Uploading	0
Done	0
Aborted by user	407

Main page

Figura 21: Resumen de estados y resultados del lote de 10.000 simulaciones (panel de administración).

A partir de estos datos, podemos sacar las siguientes conclusiones, mediante las gráficas que se verán a continuación. Primero de todo, en la siguiente figura se puede ver la proporción de tareas de éxito frente a aquellas que informaron un error de cómputo.

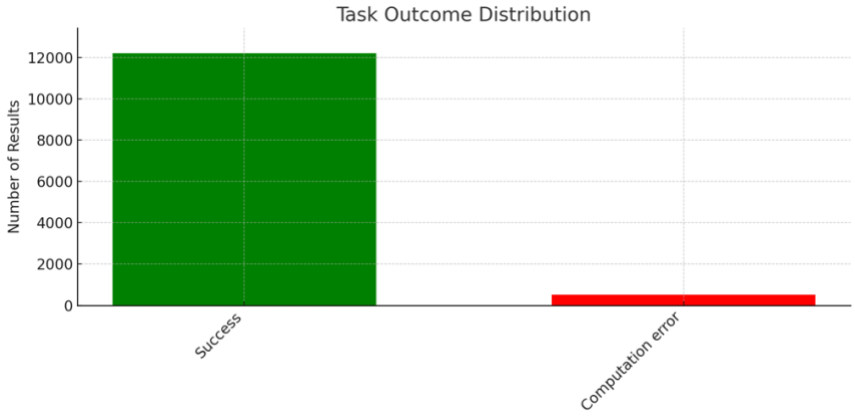


Figura 22: Task Outcome Distribution – Éxitos (verde) vs. errores de cómputo (rojo).

Como puede observarse, la gran mayoría de las tareas (12.212) terminaron correctamente, mientras que sólo 513 devolvieron un error de cómputo, lo que equivale a un 4 % del total.

Posteriormente, una vez ejecutadas, las tareas se validan cruzando salidas idénticas de dos voluntarios con plataformas idénticas (es por eso por lo que el número de tareas validas, frente a las que han terminado correctamente, se reduce prácticamente a la mitad). A continuación, podemos ver esta distribución de estados en la siguiente figura:

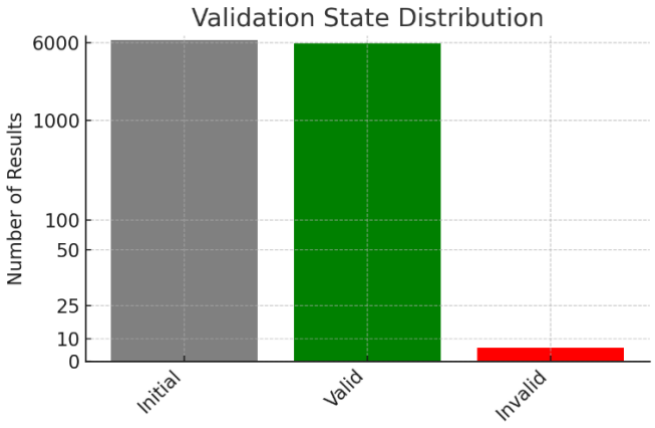


Figura 23: Validation State Distribution – Estado inicial (gris), validado (verde) e inválido (rojo).

De las 6.346 tareas que entraron en validación, 5.850 fueron marcadas como validas (el resto que no han sido validadas están esperando a su tarea gemela, para así ser validada) y únicamente 6 se declararon invalidas, estas últimas debido a problemas de apertura del archivo durante la carga del checkpoint, lo que llevo a los cálculos tomar valores no numéricos (NaN). Dado este

porcentaje tan bajo de resultados invalidas, para simulaciones tan largas, confirma que los checkpoints y la lógica de validación funcionan adecuadamente.

Por último, como se puede ver en el panel de administración, todos los errores de cómputo han ocurrido debido a un error del cliente. Dentro de estos errores, podemos diferenciar claramente dos:



Figura 24: Client Error Distribution – Errores de cómputo y tareas abortadas por el usuario.

De las 513 tareas con errores de cliente, sólo 106 corresponden a fallos internos, los cuales corresponden en su totalidad a problemas posteriores a la descarga de datos del servidor (o bien del simulador o del archivo de configuración del servidor) debido a que el archivo no se ha encontrado o se ha dañado, mientras que 407 fueron abortadas manualmente por el usuario.

7.3. Test de variabilidad de la conductancia de la corriente Ks

En esta prueba hemos preparado un lote de 10.000 simulaciones de una fibra de 1 cm (101 posiciones, para $\Delta x = 0.01\ cm$), para evaluar la variabilidad que introduce una pequeña heterogeneidad iónica. En cada simulación, G_{Ks} se asigna de forma aleatoria en cada célula dentro de un rango de ±25 % respecto a su valor de referencia. Dicho lote, se ha mandado en con dos tiempos diferentes de simulación total, el primero con 10s, y el segundo con 50s.

Las configuraciones se han generado de manera automática mediante un pequeño script, que crea, para cada posición de la fibra, un valor de G_{Ks} muestreado uniformemente en $[0{,}392 \cdot 0.75, 0{,}392 \cdot 1{,}25]$. El resultado es un conjunto homogéneo de simulaciones cuyos únicos cambios son los valores de conductancia distribuidos aleatoriamente.

Con esta prueba, se busca medir los tiempos de completado del lote en la plataforma, además de validar que la variación introducida genera dispersión en los potenciales de acción (AP) y una ligera variación en los marcadores electrofisiológicos que se postprocesan.

A continuación, se muestran los resultados obtenidos tras ejecutar las 10.000 simulaciones de ambos lotes de la prueba de variabilidad de G_{Ks}. En la siguiente figura se presentan los histogramas de los marcadores APD10, APD25, APD50, APD75 y APD90 extraídos del último pulso de cada simulación en la posición central de la fibra (posición 50). Para ello, los marcadores se extrajeron del archivo de postprocesado que llega de los ordenadores voluntarios.

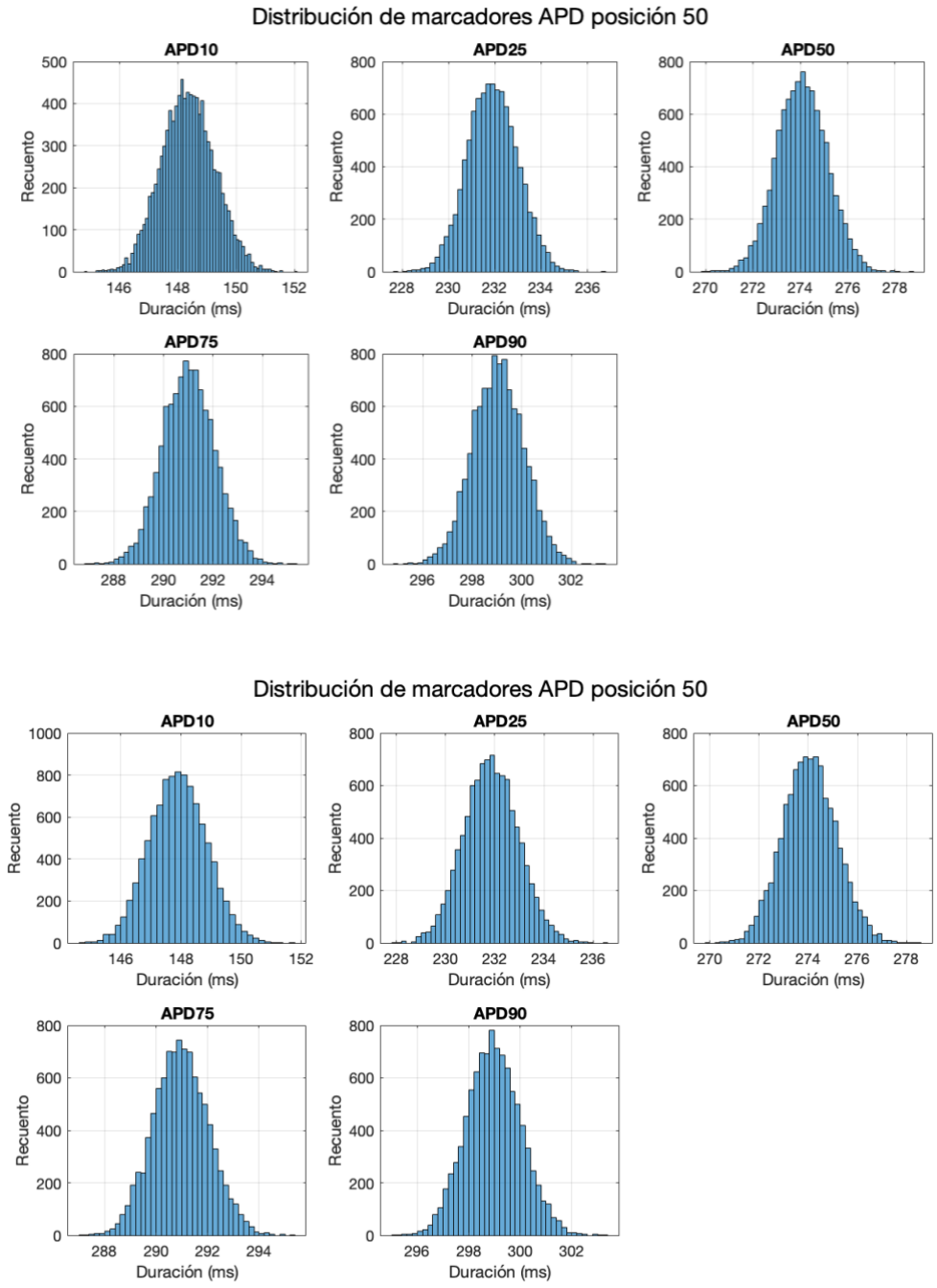


Figura 25: Histogramas de la distribución de APD10, APD25, APD50, APD75 y APD90 para la posición 50 de ambos experimentos (primera imagen prueba de 10s, segunda de 50s).

Como se puede observar, las distribuciones son aproximadamente gaussianas, centradas en los valores medios descritos en la siguiente figura:

```
APD10: mu=148.36 ms, sigma=0.91 ms   APD10: mu=147.90 ms, sigma=0.94 ms
APD25: mu=231.90 ms, sigma=1.08 ms   APD25: mu=231.85 ms, sigma=1.11 ms
APD50: mu=274.07 ms, sigma=1.06 ms   APD50: mu=274.06 ms, sigma=1.09 ms
APD75: mu=291.01 ms, sigma=1.05 ms   APD75: mu=290.97 ms, sigma=1.07 ms
APD90: mu=299.04 ms, sigma=1.03 ms   APD90: mu=298.97 ms, sigma=1.06 ms
```

Figura 26: Medias y desviaciones típicas de los histogramas (respectivamente) del APD para la posición 50.

La desviación típica (sigma) de ~1 ms en todos los marcadores muestra que la heterogeneidad de ±25 % en G_{Ks} desplaza los APD de manera detectable, pero perfectamente acotada.

A continuación, en la Figura 27 aparece la superposición de todos los potenciales de acción en la posición central de la fibra en gris claro frente a la simulación base (sin cambios) en rojo. Las variaciones de la conductancia G_{Ks}, como se podrá ver en la figura, generan variaciones en la duración de la fase de la meseta, o fase 2 del potencial de acción, haciendo esta más corta o duradera durante el potencial de acción, lo que conlleva un inicio más temprano o más tardío de la fase de la repolarización rápida, o fase 3 del potencial de acción (véase apartado 2.1.3 para la comprensión de las fases del potencial de acción). Como era esperable, ambas simulaciones arrojan resultados similares.

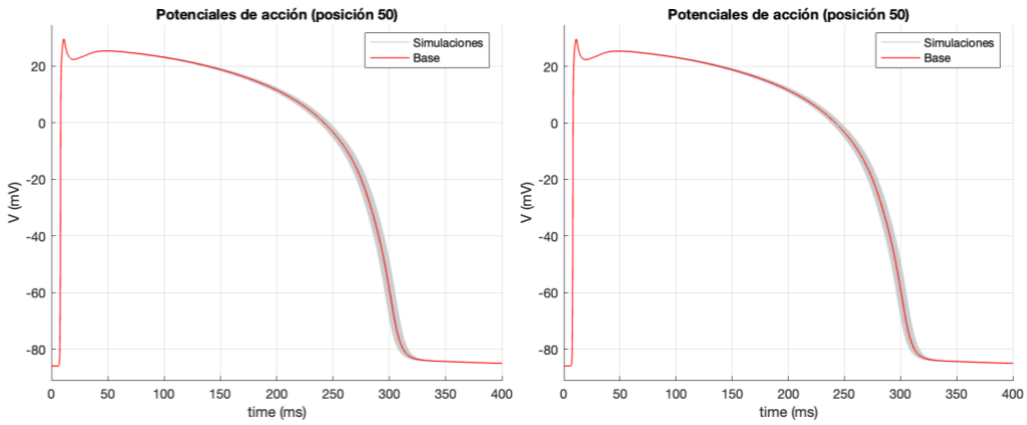


Figura 27: Potenciales de acción obtenidos de la prueba, superpuestos en la posición 50 frente a la simulación base. Primera correspondiente con la prueba de 10s, segunda con la de 50s.

Por último, la Figura 28 muestra el porcentaje de tareas completadas en función del tiempo transcurrido desde el inicio de la prueba, siendo las líneas grises los subconjuntos de simulaciones de la prueba y la curva promedio destacada en negro. Como se puede observar, el 80 % de las simulaciones se validan en torno a las 15–18h en las simulaciones de 10s y 58-62h en las de 50s, mientras que alcanzar el 100 % requiere alrededor de 6 días en ambas situaciones.

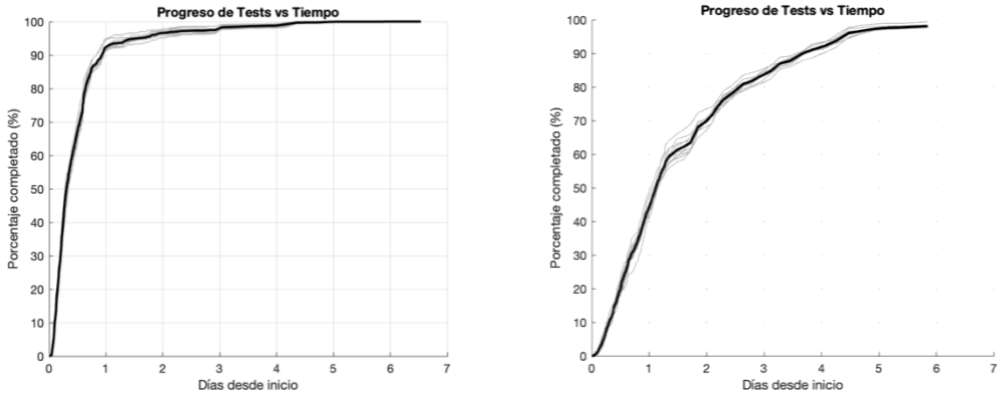


Figura 28: Gráficas de progreso de la prueba a lo largo del tiempo desde su lanzamiento hasta su fin. La primera correspondiente a las simulaciones de 10s y la segunda a las de 50s.

En la gráfica de progreso de las simulaciones de 50 segundos de duración, como se puede observar también, no llega al 100%. Esto se debe a que en el momento de la entrega del TFG aún quedó un pequeño porcentaje de tareas sin completar. Al estimular un tiempo máximo de 4 días para dar los resultados, las tareas que superan dicho límite se vuelven a enviar a voluntarios con el mismo plazo, alargando mucho el tiempo de entrega si el segundo voluntario que recibe la tarea también es lento. En DENIS@Home, existen mecanismos para terminar rápido estas pocas tareas que quedan, pero no se han utilizado en el proyecto para ver el comportamiento puro de la red.

7.4. Desviaciones a lo largo del proyecto

A lo largo del desarrollo de este proyecto, no ha sido necesario modificar los objetivos ni la metodología originalmente planteados en apartados anteriores. Gracias a todas las pruebas planteadas, se han podido se ha podido validar la funcionalidad relevante de la plataforma, corroborando así que se cumplían con los objetivos establecidos.

Sin embargo, sí se han producido varias desviaciones en la planificación que han resultado fundamentales:

- **Desarrollo de un validador propio**: Para controlar la integridad de los resultados obtenidos entre voluntarios, se implementó un validador interno capaz de comparar salidas numéricas y detectar discrepancias. Esta herramienta, no prevista en la planificación inicial, era completamente necesaria para el despliegue de la aplicación dentro del proyecto de BOINC y se descubrió mediante el análisis de requisitos para desplegar una aplicación de BOINC. Al modificarse los ficheros de salida con respecto a lo utilizado en el simulador anterior, fue necesario adaptar el validador para incluir la posibilidad de ficheros JSON.

- **Entorno de pruebas local con Docker Compose**: Con el fin de hacer pruebas menores y prevenir errores de despliegue masivo, se creó un entorno reproducible basado en Docker Compose que emula la infraestructura de DENIS@Home. Este entorno de pruebas permitió testear nuevas funcionalidades y corregir posibles fallos, antes de lanzar nuevas versiones en la plataforma real.
- **Implementación de checkpoints**: Aunque no formaba parte de los requisitos originales, se integró un sistema de puntos de control. De esta manera, las simulaciones de larga duración pueden reanudarse tras cualquier interrupción, reduciendo así el tiempo desaprovechado en hosts debido a cortes inesperados.
- **Selección de casos de prueba más concretos**: Durante la fase de diseño de experimentos, se decidió refinar los tests propuestos en la planificación inicial. En lugar de centrarnos en casos límite extremos (como el testing de casos limite o la prueba del XML), priorizamos cargas de trabajo representativas (variabilidad de conductancia y validación frente a ELVIRA) que ofrecen una evaluación más directa de la robustez y fidelidad del simulador. Así como dentro de estos tests, también se introdujo uno nuevo para comprobar que los checkpoints desarrollados funcionaban según lo esperado (tareas validadas correctamente después del guardado y carga del estado de la simulación).

Paralelamente, la retroalimentación de los voluntarios y los informes de error generados en la plataforma condujeron a sucesivas versiones beta del simulador. Cada entrega incorporó correcciones y optimizaciones hasta alcanzar la estabilidad buscada para un lanzamiento de una versión definitiva. A continuación, se enumeran las mejoras principales introducidas en cada versión:

- **beta_0.01**: Primera versión que se lanzó a la red de voluntarios del simulador de fibras cardiacas heterogéneas. Esta versión, muy simple y estable, ya era capaz de resolver cálculos con propagación del estímulo aplicado, además de aplicar variaciones a células concretas de la fibra (variaciones intrapaciente).
- **beta_0.02**: Derivada de un problema en la firma del binario para arquitecturas de Linux trabajando sobre ARM (aarch64-unknown-linux-gnu). No hubo cambios en el código entre esta nueva versión y la anterior. Con el lanzamiento de esta versión, se solucionó el problema de la firma.
- **beta_0.03**: Derivada de un problema de optimización para compilados Windows (explicado en el Anexo IV), que hacía que las simulaciones de Windows frente a la de otras plataformas tardara hasta 4 veces más. Con la salida de esta versión, se resolvió este problema de compilación.

- **beta_0.04**: Primera versión con checkpoints del simulador, además de implementar el cambio de parámetros para todas las células globalmente dentro de la fibra (variaciones interpaciente), y la posibilidad de establecer un coeficiente de difusión deseado.
- **beta_0.05**: Se modifico el output que generaban tanto los checkpoints como el simulador al finalizar su ejecución en el archivo stderr.txt, puesto que para simulaciones muy largas no se podía apreciar donde se habían generado problemas de carga en los checkpoints.

8. Conclusiones

8.1. Grado de cumplimiento de los objetivos planteados

El proyecto se planteó con el objetivo principal de adaptar la plataforma DENIS para permitir la simulación de fibras heterogéneas y, junto a ello, cuatro objetivos secundarios: desarrollar un simulador 1D, incorporar variabilidad inter- e intrapaciente, descentralizar el postprocesado de marcadores electrofisiológicos y garantizar la portabilidad multiplataforma. Tras el desarrollo, lanzamiento de nuevas versiones en la plataforma y las pruebas ejecutadas, puede afirmarse que todos estos objetivos se han alcanzado de manera satisfactoria.

Desde el punto de vista funcional, el simulador 1D se ha completado con éxito y se integra sin problemas en la infraestructura de BOINC existente. Este simulador es capaz de resolver fibras heterogéneas gracias a los métodos numéricos implementados y a la correcta ampliación de la plataforma y la nueva lógica implementada para la propagación del estímulo, además de generar un archivo de salida con las variables deseadas.

Con respecto a la heterogeneidad, gracias a la prueba de la variación de la conductancia hemos podido observar cómo efectivamente se ha aplicado correctamente la variabilidad interpaciente (parámetros globales) y la como la intrapaciente (parámetros por célula), ya que, efectivamente, se han producido ligeras variaciones en los potenciales de acción obtenidos de la prueba.

Asimismo, el sistema de postprocesado distribuido ha funcionado según lo previsto: cada nodo voluntario calcula los marcadores (APD10, APD25, APD50, APD75, APD90…) y devuelve un JSON resumido, aliviando la carga del servidor central y garantizando datos para el posterior análisis.

Además, a lo largo del lanzamiento de nuevas versiones, se han ido generando los compilados a través de los scripts de compilación desarrollados con dicho propósito, lo cual ha agilizado bastante el proceso de despliegue de nuevas versiones de la aplicación en la plataforma. Asimismo, no se han reportado fallos de ejecución por parte del binario generado por los voluntarios, ya que los errores de ejecución descubiertos han sido por cuestiones ajenas a la aplicación. Es por esto, que también se cumple el objetivo de compilación multiplataforma.

8.2. Adaptaciones realizadas y aprendizajes durante el desarrollo

A pesar de que los objetivos y la metodología iniciales se mantuvieron inalterados, la incorporación de tres mejoras no previstas aportó un valor añadido sustancial al proyecto. Estas adaptaciones, lejos de desviar recursos, optimizaron la calidad de los resultados, agilizaron los ciclos de prueba y reforzaron la robustez de la plataforma.

La implantación de un validador interno y un entorno de pruebas local con Docker Compose respondieron a la necesidad de garantizar la consistencia de los datos y la reproducibilidad de las ejecuciones antes de enviarlas a los voluntarios. El validador, exigido por la infraestructura BOINC, permitió detectar discrepancias numéricas, de tal forma que si dos simulaciones tienen valores muy diferentes (error relativo >0.1%) este iba a indicar que no es válida. Por su parte, el entorno de pruebas de Docker aceleró la validación de nuevas versiones y facilitó la detección de posibles errores, sin tener que pasar por el entorno real para ello.

La integración de checkpoints supuso un avance decisivo para las simulaciones de larga duración. Al poder interrumpir y reanudar cálculos (bien intencionadamente o no) sin pérdida de información, se redujo el tiempo desaprovechado en hosts voluntarios y se aumentó la tasa de tareas completadas con éxito.

En el ámbito de la experimentación, la selección estratégica de casos de prueba ayudo a darle un enfoque más correcto de cara a la comprobación de los objetivos que se habían establecido. Gracias a esta decisión se obtuvieron así resultados que verificaban el verdadero funcionamiento de la plataforma, ya que los casos muy específicos, como bien dice su nombre, son condiciones que se cumplen pocas veces, y las pruebas que se estaban buscando tenían que ser más relevantes en cuanto a los resultados obtenidos, en lugar de como se iba a comportar la plataforma ante casos muy específicos (pruebas del XML, casos límite de simulaciones…).

8.3. Posibles mejoras y futuras ampliaciones

8.3.1. Pruebas adicionales a medio/largo plazo

Además de las pruebas propuestas en el apartado 7, también resulta esencial validar el comportamiento de DENIS en simulaciones prolongadas, donde los plazos y la gestión de relanzamientos cobran especial importancia. Aunque estas pruebas propuestas buscan una evaluación muy profunda, dado al marco de realización del trabajo de fin de grado y el tiempo que llevaría la ejecución de estas pruebas, resultaría inviable realizarlas a corto plazo. Las pruebas consistirían en lo siguiente:

- **Duración física de la simulación (t_final)**: Se analiza si, al aumentar el tiempo de simulación (en milisegundos), el tiempo total de completado de un lote de fibras crece de forma proporcional o si la infraestructura amortigua ese incremento.
- **Plazo máximo de entrega (deadline)**: Se comprueba cómo influyen distintos plazos límite en la rapidez con que los voluntarios pueden entregar resultados, y si existe un umbral mínimo de deadline por debajo del cual la red no alcanza a completar todas las tareas.

- **Tamaño del lote de simulaciones**: Se investiga hasta qué punto el número de fibras enviadas simultáneamente (miles de WU) afecta al tiempo de respuesta global, identificando posibles puntos de saturación en la cola de trabajo.

Como se puede ver, son pruebas que plantean una evaluación muy robusta de las características que nos ofrece la plataforma de BOINC. Estas pruebas requerirían incluso de volver a lanzar repetidas veces los lotes de simulaciones preparados para cada una, para testear así posibles desviaciones; es por eso por lo que requerirían de varios meses para su completa realización. Por lo que, debido a esto, a corto plazo resultarían inviables, aunque para medio y largo plazo serian ideales.

8.3.2. Mejoras con respecto al simulador

A corto plazo, la actualización del cliente que provee las librerías de compilación desde la versión 7.22.0 (mayo 2023) a la actual 8.0.4 (diciembre 2024) también aportaría beneficios: mejor soporte de lenguajes y librerías modernas, optimizaciones automáticas de código y detección temprana de incompatibilidades. Un cliente actualizado reduce el riesgo de fallos de build en plataformas emergentes, o incluso actualizaciones de las ya existentes, y suele incluir parches de seguridad, lo que fortalece la fiabilidad del proceso de distribución de ejecutables.

A corto/medio plazo, ampliar el repertorio de modelos celulares soportados enriquecería significativamente la versatilidad de DENIS. Actualmente la plataforma incluye veinte modelos distintos; incorporar unos cuantos adicionales permitiría abordar nuevos estudios, que podrían ser interesantes para abrir puertas hacia nuevos proyectos de investigación.

En paralelo, la implementación de métodos de resolución numérica extra mejoraría el compromiso entre precisión y tiempo de cómputo. Aunque hoy se ofrecen Forward Euler y Runge–Kutta de segundo y cuarto orden, se podría añadir un integrador adaptativo (por ejemplo, Runge–Kutta–Fehlberg) o métodos de órdenes superiores para experimentos donde la exactitud de la fase de repolarización sea crítica. Esta mejora nos permitiría seleccionar el esquema más adecuado según las características de cada estudio.

A muy largo plazo, la evolución hacia simulaciones en 2D y 3D representaría un gran salto para la electrofisiología en computación voluntaria: analizar la propagación en bloques de tejido ofrecería una visión más realista de fenómenos como la fibrilación o la formación de reentradas. No obstante, esta ampliación exigirá rediseñar de nuevo el simulador, adaptando así este de nuevo para soportar este tipo de simulaciones, además de asegurar que la infraestructura BOINC puede gestionar cargas de trabajo con requisitos de memoria y CPU mucho más elevados.

8.3.3. Mejoras con respecto al servidor de BOINC

Para medio plazo, conviene migrar la infraestructura a la nube. Alojar el servidor de BOINC en AWS, Azure o GCP permitiría escalar dinámicamente CPU, memoria y almacenamiento según la demanda de trabajo voluntario, mejorar la disponibilidad global y desplegar balanceadores de carga geográficamente distribuidos. Esta transición también simplificaría la gestión de backups y recuperaciones ante fallos.

A largo plazo, si se prefiere conservar hardware propio, resulta recomendable renovar el servidor local de la USJ, que lleva más de diez años en funcionamiento (desde febrero de 2015). Un equipo moderno con discos NVMe, procesadores multinúcleo recientes y mayor capacidad de virtualización ofrecerá un rendimiento superior, especialmente bajo picos de carga intensiva. Esta inversión garantizaría que la plataforma siga siendo robusta para la cada vez mayor demanda de computación de los programas, y capaz de atender requisitos crecientes en proyectos futuros.

8.4. Reflexión final

En este proyecto he comprobado de primera mano lo que implica diseñar y desplegar una aplicación de simulación distribuida a gran escala. El trabajo ha movilizado desde la elección de métodos numéricos hasta la gestión de miles de voluntarios, pasando por la creación de entornos de prueba y sistemas de validación.

He aprendido a adaptar rápidamente los planes cuando surgían imprevistos, como al desarrollar el validador interno o al integrar puntos de control en las simulaciones largas. Estas decisiones, aunque no formaran parte de la propuesta inicial, resultaron fundamentales para garantizar la fiabilidad de los resultados y obtener un mejor producto final.

La experiencia con Docker Compose y las actualizaciones de compilación me enseñó la importancia de contar con entornos reproducibles: ahorran tiempo y reducen el riesgo de errores en producción. Además, coordinar y analizar lotes masivos de simulaciones me ayudo a mejorar mis habilidades estadísticas y me permitió interpretar de forma rigurosa la variabilidad de los resultados que se pueden obtener con pequeñas variaciones.

El uso de computación distribuida también me ha ayudado a conocer la importancia de los foros, donde escuchas en primera mano a los voluntarios de tu proyecto: cuales son los problemas que hay con nuevas versiones, posibles recomendaciones, o incluso las ayudas que te proporcionan.

En definitiva, este TFG no solo ha servido para cumplir los objetivos científicos y técnicos planteados, sino que ha sido, sobre todo, una oportunidad para crecer profesionalmente. Me llevo la confianza de saber que puedo liderar proyectos complejos, gestionar infraestructuras distribuidas y extraer conclusiones sólidas a partir de grandes volúmenes de datos.

9. Bibliografía

[1]. Organización Mundial de la Salud. (2021). *Enfermedades cardiovasculares*. https://www.who.int/es/health-topics/cardiovascular-diseases#tab=tab_1

[2]. Bover, Ramon (2017). *Estancia media, mortalidad intrahospitalaria y reingresos en pacientes hospitalizados por insuficiencia cardiaca en España*. Universidad Complutense de Madrid. https://hdl.handle.net/20.500.14352/21798

[3]. Noble, D.; Garny, A. & Noble, P. J. (2012). How the Hodgkin–Huxley equations inspired the Cardiac Physiome Project. *The Journal of Physiology*, *590*(11), 2613–2628. https://doi.org/10.1113/JPHYSIOL.2011.224238

[4]. Ni, H.; Morotti, S. & Grandi, E. (2018). A heart for diversity: Simulating variability in cardiac arrhythmia research. *Frontiers in Physiology*, *9*(JUL), 391359. https://doi.org/10.3389/FPHYS.2018.00958/BIBTEX

[5]. Corral-Acero, J., et al. (2020). The 'Digital Twin' to enable the vision of precision cardiology. *European Heart Journal*, *41*(48), 4556–4564. https://doi.org/10.1093/EURHEARTJ/EHAA159

[6]. Malmivuo, J. & Plonsey, R. (1995). *Bioelectromagnetism - Principles and Applications of Bioelectric and Biomagnetic Fields.* Oxford University Press, New York.

[7]. Anderson, D. P. (2020). BOINC: A Platform for Volunteer Computing. *Journal of Grid Computing*, *18*(1), 99–122. https://doi.org/10.1007/S10723-019-09497-9/METRICS

[8]. Monasterio, V.; Castro-Mur, J. & Carro, J. (2018). DENIS: Solving cardiac electrophysiological simulations with volunteer computing. *PLOS ONE*, *13*(10), e0205568. https://doi.org/10.1371/JOURNAL.PONE.0205568

[9]. Carro, J.; Monasterio, V. & Castro, J. (2018, May 18). *DENIS Project*. https://denis.usj.es/

[10]. Cuellar, A., et al. (2015). The CellML 1.1 Specification. *Journal of Integrative Bioinformatics*, *12*(2), 4–85. https://doi.org/10.1515/JIB-2015-259

[11]. Kléber, A. G. & Rudy, Y. (2004). Basic Mechanisms of Cardiac Impulse Propagation and Associated Arrhythmias. *Physiological Reviews*, *84*(2), 431–488. https://doi.org/10.1152/PHYSREV.00025.2003/ASSET/IMAGES/LARGE/Z9J00204029400 41.JPEG

[12]. Shih, H.-T., et al. (1994). Anatomy of the action potential in the heart. *Texas Heart Institute Journal*, *21*(1), 30. https://pmc.ncbi.nlm.nih.gov/articles/PMC325129/

[13]. Hodgkin, A. L. & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, *117*(4), 500–544. https://doi.org/10.1113/JPHYSIOL.1952.SP004764

[14]. Noble, D. (1962). A modification of the Hodgkin—Huxley equations applicable to Purkinje fibre action and pacemaker potentials. *The Journal of Physiology*, *160*(2), 317–352. https://doi.org/10.1113/JPHYSIOL.1962.SP006849

[15]. Beeler, G. W. & Reuter, H. (1977). Reconstruction of the action potential of ventricular myocardial fibres. *The Journal of Physiology*, *268*(1), 177–210. https://doi.org/10.1113/JPHYSIOL.1977.SP011853

[16]. Luo, C. H. & Rudy, Y. (1991). A model of the ventricular cardiac action potential. Depolarization, repolarization, and their interaction. *Circulation Research*, *68*(6), 1501–1526. https://doi.org/10.1161/01.RES.68.6.1501

[17]. Luo, C. H. & Rudy, Y. (1994). A dynamic model of the cardiac ventricular action potential. I. Simulations of ionic currents and concentration changes. *Circulation Research*, *74*(6), 1071–1096. https://doi.org/10.1161/01.RES.74.6.1071

[18]. ten Tusscher, K. H. W. J. & Panfilov, A. V. (2006). Alternans and spiral breakup in a human ventricular tissue model. *American Journal of Physiology - Heart and Circulatory Physiology*, *291*(3), 1088–1100. https://doi.org/10.1152/AJPHEART.00109.2006

[19]. Grandi, E., et al. (2009). A Novel Computational Model of the Human Ventricular Action Potential and Ca transient. *Biophysical Journal*, *96*(3), 664a–665a. https://doi.org/10.1016/j.bpj.2008.12.3513

[20]. O'Hara, T., et al. (2011). Simulation of the Undiseased Human Cardiac Ventricular Action Potential: Model Formulation and Experimental Validation. *PLOS Computational Biology*, *7*(5), e1002061. https://doi.org/10.1371/JOURNAL.PCBI.1002061

[21]. Tomek, J., et al. (2019). Development, calibration, and validation of a novel human ventricular myocyte model in health, disease, and drug block. *ELife*, *8*. https://doi.org/10.7554/ELIFE.48890

[22]. Anderson, D. P. (2003). *Public Computing: Reconnecting People to Science*. http://boinc.berkeley.edu/talks/madrid_03/madrid.html

[23]. Anderson, D. P. (2004). BOINC: A system for public-resource computing and storage. *Proceedings - IEEE/ACM International Workshop on Grid Computing*, 4–10. https://doi.org/10.1109/GRID.2004.14

[24]. Anderson, D. P., et al. (2002). SETI@Home: An Experiment in Public-Resource Computing. *Communications of the ACM, Vol. 45 No. 11*, 56–61.

[25]. Das, R., et al. (2007). Structure prediction for CASP7 targets using extensive all-atom refinement with Rosetta@home. *Proteins*, *69 Suppl 8*(SUPPL. 8), 118–128. https://doi.org/10.1002/PROT.21636

[26]. Giorgino, T.; Harvey, M. J. & de Fabritiis, G. (2010). Distributed computing as a virtual supercomputer: Tools to run and manage large-scale BOINC simulations. *Computer Physics Communications, 181(8)*, 1402–1409. https://doi.org/10.1016/J.CPC.2010.04.007

[27]. Buch, I., et al. (2010). High-throughput all-atom molecular dynamics simulations using distributed computing. *Journal of Chemical Information and Modeling, 50(3)*, 397–403. https://doi.org/10.1021/CI900455R

[28]. Harvey, M. J.; Giupponi, G. & de Fabritiis, G. (2009). ACEMD: Accelerating Biomolecular Dynamics in the Microsecond Time Scale. *Journal of Chemical Theory and Computation, 5(6)*, 1632–1639. https://doi.org/10.1021/CT9000685

[29]. Heidenreich, E. A., et al. (2010). Adaptive macro finite elements for the numerical solution of monodomain equations in cardiac electrophysiology. *Annals of Biomedical Engineering, 38(7)*, 2331–2345.

[30]. Niederer, S. A., et al. (2011). Verification of cardiac tissue electrophysiology simulators using an N-version benchmark. *Philosophical Transactions. Series A, Mathematical, Physical, and Engineering Sciences, 369(1954)*, 4331–4351.

[31]. Pitt-Francis, J., et al. (2009). Chaste: A test-driven approach to software development for biological modelling. *Computer Physics Communications*, *180*(12), 2452–2471. https://doi.org/10.1016/j.cpc.2009.07.019

[32]. Pathmanathan, P., et al. (2010). A numerical guide to the solution of the bidomain equations of cardiac electrophysiology. In *Progress in Biophysics and Molecular Biology* (Vol. 102, Issues 2–3, pp. 136–155). https://doi.org/10.1016/j.pbiomolbio.2010.05.006

[33]. Plank, G., et al. (2021). The openCARP simulation environment for cardiac electrophysiology. *Computer Methods and Programs in Biomedicine*, *208*. https://doi.org/10.1016/j.cmpb.2021.106223

[34]. Schwaber, K. & Sutherland, J. (2020). *The Scrum Guide*. https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

[35]. *Disen, V. (2010) A successful Git branching model» nvie.com*. (2010). https://nvie.com/posts/a-successful-git-branching-model/

[36]. Keener, J. & Sneyd, J. (1998). *Mathematical Physiology*. *8*. https://doi.org/10.1007/B98841

[37]. Taggart, P., et al. (2000). Inhomogeneous transmural conduction during early ischaemia in patients with coronary artery disease. *Journal of Molecular and Cellular Cardiology*, *32*(4), 621–630. https://doi.org/10.1006/jmcc.2000.1105

[38]. *Home · open-source-parsers/jsoncpp Wiki*. (2017). Obtenido el 27 de mayo de 2025, de https://github.com/open-source-parsers/jsoncpp/wiki

[39]. *Create a BOINC server (cookbook) · BOINC/boinc Wiki*. (2025). Obtenido el 27 de mayo de 2025, de https://github.com/BOINC/boinc/wiki/Create-a-BOINC-server-(cookbook)

[40]. *CodeSigning – BOINC*. (2014). Obtenido el 27 de mayo de 2025, de https://boinc.berkeley.edu/trac/wiki/CodeSigning

[41]. *JobSubmission – BOINC*. (2019). Obtenido el 29 de mayo de 2025, de https://boinc.berkeley.edu/trac/wiki/JobSubmission

[42]. *BOE-A-2023-17238 Resolución de 13 de julio de 2023, de la Dirección General de Trabajo, por la que se registra y publica el XVIII Convenio colectivo estatal de empresas de consultoría, tecnologías de la información y estudios de mercado y de la opinión pública.* (2023). Obtenido el 3 de junio de 2025, de https://www.boe.es/diario_boe/txt.php?id=BOE-A-2023-17238

[43]. *Seguridad Social: Cotización / Recaudación de Trabajadores*. (2025). Obtenido el 3 de junio de 2025, de https://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/10721/10957/583

[44]. Cooper, J., Spiteri, R. J., & Mirams, G. R. (2015). Cellular cardiac electrophysiology modeling with Chaste and CellML. *Frontiers in Physiology*, *6*(JAN), 112900. https://doi.org/10.3389/FPHYS.2014.00511/BIBTEX

[45]. Rush, S. & Larsen, H., "A practical algorithm for solving dynamic membrane equations", *IEEE Trans. Biomed. Eng.*, vol. BME-25, no. 4, pp. 389–392, 1978. https://doi.org/10.1063/5.0127713

10. Anexos

ANEXO I – Propuesta del Proyecto

Nombre alumno:	Iván Royo Gutiérrez
Titulación:	Grado en Ingeniería Informática
Curso académico:	2024-2025

1. TÍTULO DEL PROYECTO

Computación voluntaria para la simulación de tejidos cardiacos heterogéneos

2. DESCRIPCIÓN Y JUSTIFICACIÓN DEL TEMA A TRATAR

El proyecto de computación voluntaria DENIS (Distributed Computing, Electrophysiological Models, Networking Collaboration, In Silico Research, Sharing Knowledge), accesible en la web https://denis.usj.es y desarrollado en la Universidad San Jorge, nació en febrero de 2015 con el propósito de crear una red de ordenadores voluntarios para ayudar a investigadores en el ámbito de la electrofisiología cardiaca. El proyecto se hizo público en abril de 2015 y desde entonces se han sumado a él más de 5400 voluntarios de todo el mundo que comparten casi 35.000 ordenadores para realizar simulaciones de la actividad eléctrica del corazón. Actualmente la plataforma DENIS está preparada para simular la actividad eléctrica de las células del corazón de manera aislada. Gracias al gran número de voluntarios colaboradores, la red es ya capaz de realizar 60.000 simulaciones de una hora de actividad de una célula cardiaca al día.

En la actualidad se pueden realizar únicamente simulaciones de una célula cardiaca, pero para llevar a cabo simulaciones multiescala se debe seguir haciendo crecer la plataforma.

3. OBJETIVOS DEL PROYECTO

El objetivo principal del proyecto es dar un primer paso para adaptar la plataforma DENIS a la simulación multiescala y permitir así la ejecución de fibras heterogéneas. Para ello se establecen los siguientes objetivos secundarios:

- **Desarrollar un nuevo simulador para DENIS que resuelva simulaciones 1D**: en una primera aproximación, estas células deben ser todas del mismo tipo y utilizar el mismo modelo.

- **Simulación heterogénea**: dentro de las fibras a simular, se podrán modificar los diferentes parámetros del modelo para todas las células (interpaciente) o entre las diferentes células de la simulación (intrapaciente).
- **Post-procesado en los ordenadores voluntarios**: Para reducir el flujo de información y evitar que el servidor se sobrecargue de información, al igual que en el simulador 0D, todos aquellos marcadores que sea posible se calcularán antes de que las simulaciones se devuelvan. Se deberían mantener al menos los actuales: APD_{90}, APD_{50}, Triangulación, etc.
- **Simulador multiplataforma**: El proyecto debe seguir siendo multiplataforma y tiene que poder compilarse el mismo código (o con pequeñas variaciones) en Windows, Linux y MacOS. En la medida de lo posible, se automatizará esta tarea.

4. DOCENTE QUE RESPALDA LA PROPUESTA

El proyecto cuenta con el aval de Jesús Carro Fernández.

ANEXO II – Actas de Reunión

Reunión 1

REUNIÓN:	**Presentación Propuesta de TFG**

Fecha:	17/09/2024		
Hora comienzo:	9:30	**Hora finalización:**	10:30
Lugar:	Universidad San Jorge		
Elabora acta:	Iván Royo Gutiérrez		
Convocados:	Iván Royo Gutiérrez, Jesús Carro Fernández		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Disculpas por ausencia** No procede.	--
2	**Aprobar última acta** No procede.	--
3	**Asuntos pendientes última acta** No procede.	--
4	**Redacción de borrador sobre la propuesta del TFG** Iván Royo Gutiérrez y Jesús Carro Fernández redactan un borrador sobre la propuesta con las ideas que habían acordado.	001
5	**Revisión de la propuesta** Ambas partes se comprometen a revisar la propuesta, con la finalidad de mejorar y llegar a una versión final.	002
6	**Creación de la versión definitiva de la propuesta** En base a la revisión, Iván Royo Gutiérrez se encarga de elaborar la versión final de la propuesta.	003
7	**Otros asuntos** No se trataron otros asuntos relevantes.	--
8	**Próxima reunión** Se discutirá acerca de la planificación inicial para empezar el proyecto y acerca del material necesario para su desarrollo.	--

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Redactar el borrador de la propuesta del TFG	17/09/24	Iván Royo Gutiérrez y Jesús Carro Fernández
002	Revisión del borrador de la propuesta	15/10/24	Iván Royo Gutiérrez y Jesús Carro Fernández
003	Elaboración de una versión definitiva, procedente de las revisiones realizadas.	18/10/24	Iván Royo Gutiérrez

Reunión 2

REUNIÓN:	**Planificación Inicial del Trabajo y Material Necesario**

Fecha:	12/11/2024		
Hora comienzo:	9:30	**Hora finalización:**	10:30
Lugar:	Universidad San Jorge		
Elabora acta:	Iván Royo Gutiérrez		
Convocados:	Iván Royo Gutiérrez, Jesús Carro Fernández		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Disculpas por ausencia** No procede	--
2	**Aprobar última acta** Se aprueba el acta de la reunión del 17 de septiembre de 2024	--
3	**Asuntos pendientes última acta** No procede.	--
4	**Planteamiento inicial de como trabajar** Se decidió usar CLion como IDE para la programación del proyecto, debido a que el manejo del proyecto se gestiona con CMake.	--
5	**Entendimiento de la infraestructura existente** Jesús Carro Fernández destacó la importancia de entender cómo funciona Denis en su versión 0D a bajo nivel	001
6	**Entendimiento de la plataforma de computación voluntaria** Así mismo, Jesús Carro Fernández recalca que también es muy importante comprender la plataforma de computación voluntaria BOINC	002
7	**Desarrollo de un archivo de configuración para Denis Fiber** Una vez realizadas las fases de comprensión, Jesús Carro Fernández comentó que sería interesante empezar por desarrollar un nuevo fichero de configuración y ser capaz de leerlo desde código	003 004
8	**Otros asuntos** Jesús Carro Fernández comenta que me facilitará documentación relevante y acceso al código de la plataforma.	--
9	**Próxima reunión** Se discutirán los primeros pasos en la expansión de las simulaciones y se discutirá como abordar la memoria del TFG.	--

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Comprensión básica de la plataforma Denis Myocyte	02/12/24	Iván Royo Gutiérrez
002	Comprensión de la infraestructura de BOINC	16/12/24	Iván Royo Gutiérrez
003	Desarrollo de un archivo de configuración para el simulador	23/12/24	Iván Royo Gutiérrez
004	Desarrollo del código encargado de leer el fichero de configuración	06/01/25	Iván Royo Gutiérrez

Reunión 3

REUNIÓN:	**Expansión de Simulaciones y Estructuración de la Memoria**

Fecha:	21/01/2025		
Hora comienzo:	9:45	**Hora finalización:**	11:00
Lugar:	Universidad San Jorge		
Elabora acta:	Iván Royo Gutiérrez		
Convocados:	Iván Royo Gutiérrez, Jesús Carro Fernández		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Disculpas por ausencia** No procede.	--
2	**Aprobar última acta** Se aprueba el acta de la reunión del 12 de noviembre de 2024	--
3	**Asuntos pendientes última acta** No procede.	--
4	**Revisión del código de lectura de la configuración** Jesús Carro Fernández se compromete a revisar el código realizado para la lectura del archivo de configuración	001
5	**Modificación del output y postprocesado en ordenadores voluntarios** Jesús Carro Fernández recalca en adaptar el output a la nueva estructura y posteriormente cambiar el postprocesado para que sea un JSON en lugar de una matriz	002 003
6	**Modificación de la simulación para simular n células** Jesús Carro Fernández comenta que primero trate de hacer que se simulen n células antes de introducir propagación	004
7	**Estructuración de los apartados de la memoria** Iván Royo Gutiérrez se encargará de estructurar los apartados de la memoria, para que posteriormente Jesús Carro Fernández lo revise	005
8	**Otros asuntos** No se trataron otros asuntos importantes.	--
9	**Próxima reunión** Se discutirá sobre cómo se debe de propagar un estímulo a lo largo de la fibra cardiaca	--

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Revisión del código de lectura de la configuración	28/01/25	Jesús Carro Fernández
002	Adaptación de la clase output para las simulaciones	27/01/25	Iván Royo Gutiérrez
003	Adaptación a la nueva estructura de la clase postprocesado	17/02/25	Iván Royo Gutiérrez
004	Expansión de la simulación para simular múltiples células	03/03/25	Iván Royo Gutiérrez
005	Estructurar y definir los apartados de la memoria	27/01/25	Iván Royo Gutiérrez

Reunión 4

REUNIÓN:	**Propagación del Estímulo en la Fibra Cardiaca**

Fecha:	25/02/2025		
Hora comienzo:	9:45	**Hora finalización:**	11:30
Lugar:	Universidad San Jorge		
Elabora acta:	Iván Royo Gutiérrez		
Convocados:	Iván Royo Gutiérrez, Jesús Carro Fernández		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Disculpas por ausencia** No procede.	--
2	**Aprobar última acta** Se aprueba el acta de la reunión del 21 de enero de 2025.	--
3	**Asuntos pendientes última acta** No procede.	--
4	**Revisión del código del proyecto** Jesús Carro Fernández se compromete a mirar el código del proyecto avanzado hasta la fecha.	001
5	**Concepto de propagación del estimulo** Se revisa por ambas partes como se calcula el voltaje de una célula con la propagación de un estímulo, de tal forma que estimulando una única célula el impulso recorra todo el tejido.	--
6	**Inclusión de las fórmulas encargadas de propagar el estímulo en el código** Una vez comprendida la teoría, Iván Royo Gutiérrez analizará como se pueden incluir esas formulas en el código para que ocurra la propagación.	002
7	**Otros asuntos** No se trataron otros asuntos importantes.	--
8	**Próxima reunión** Se comentará como crear un proyecto de BOINC y se comentará la primera revisión de la memoria.	--

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Revisión del código del proyecto	11/03/25	Jesús Carro Fernández
002	Realización del código encargado de la propagación del estímulo	10/03/25	Iván Royo Gutiérrez

Reunión 5

REUNIÓN: Creación de Proyecto de BOINC y Revisión de la Memoria

Fecha:	11/03/2025		
Hora comienzo:	9:45	**Hora finalización:**	10:45
Lugar:	Universidad San Jorge		
Elabora acta:	Iván Royo Gutiérrez		
Convocados:	Iván Royo Gutiérrez, Jesús Carro Fernández		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Disculpas por ausencia** No procede.	--
2	**Aprobar última acta** Se aprueba el acta de la reunión del 25 de febrero de 2025.	--
3	**Asuntos pendientes última acta** No procede.	--
4	**Revisión de la memoria** Jesús Carro Fernández se compromete a hacer dos revisiones de la memoria consecutivas, una incluyendo los apartados de Introducción y Estado del Arte, y otra con todo al completo para la entrega intermedia	001 002
5	**Análisis de requisitos para la creación de un proyecto de BOINC** Jesús Carro Fernández comenta la necesidad de saber cómo se crea un proyecto de BOINC, externo al que hay en la universidad, para así usarlo como entorno de pruebas para el despliegue de nuevas aplicaciones en el proyecto.	003
6	**Creación de la aplicación Denis Fiber dentro del entorno de pruebas** Jesús Carro Fernández comenta también que cuando se despliegue el entorno de pruebas, se creara también la aplicación del binario para ver cómo se comporta con la infraestructura de BOINC.	004
7	**Otros asuntos** No se trataron otros asuntos importantes.	--
8	**Próxima reunión** Se hablará sobre los scripts de compilación del proyecto, y se empezará a hablar sobre la versión definitiva de la memoria.	--

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Revisión de la memoria del proyecto (Hasta el estado del arte)	24/03/25	Jesús Carro Fernández
002	Revisión de la memoria del proyecto para su entrega intermedia	08/04/25	Jesús Carro Fernández
003	Investigar cómo crear y crear un proyecto de BOINC	17/03/25	Iván Royo Gutiérrez
004	Crear una aplicación nueva dentro del proyecto, que ejecute el programa de Denis Fiber	31/03/25	Iván Royo Gutiérrez

Reunión 6

REUNIÓN:	**Requisitos adicionales de BOINC y Despliegue de la Primera Versión**

Fecha:	08/04/2025		
Hora comienzo:	9:45	**Hora finalización:**	10:45
Lugar:	Universidad San Jorge		
Elabora acta:	Iván Royo Gutiérrez		
Convocados:	Iván Royo Gutiérrez, Jesús Carro Fernández		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Disculpas por ausencia** No procede.	--
2	**Aprobar última acta** Se aprueba el acta de la reunión del 11 de marzo de 2025.	--
3	**Asuntos pendientes última acta** No procede.	--
4	**Validador y Asimilador de la aplicación** Durante la búsqueda de requisitos para la creación de un proyecto de BOINC, se descubrió que las aplicaciones nuevas requieren de un validador y de un asimilador. Iván Royo Gutiérrez se lo comento a Jesús Carro Fernández, acordando así desarrollar un validador propio, mientras que para el asimilador se usaría el que trae BOINC por defecto.	001
5	**Desarrollo de Scripts de Compilación** Jesús Carro Fernández comento la relevancia de desarrollar unos scripts de compilación, utilizando Docker para la compilación de plataformas Linux y Windows, para así poder compilar todo (en la medida de lo posible), en la menor cantidad de pasos posibles. Además, dichos scripts podrían preparar ya la estructura requerida por BOINC.	002
6	**Redacción de la parte correspondiente a la Entrega Final de la Memoria** Jesús Carro Fernández sugirió ir escribiendo la parte correspondiente a la Implementación en la memoria, sobre lo que se había hecho hasta ahora del simulador.	003
7	**Despliegue de la aplicación DENIS Fiber reducido a voluntarios** Jesús Carro Fernández comenta, que cuando se consiga compilar los binarios para las plataformas de Windows, MacOS (Intel y ARM) y Linux (Intel y ARM), si pasa una pequeña prueba en el entorno de pruebas desarrollado, se podría iniciar su despliegue a la plataforma real.	004
8	**Otros asuntos** Se comentaron la estructura de los puntos de control (checkpoints), pero se acordó que primero debía de pasar por el despliegue reducido para iniciar con su implementación.	--
9	**Próxima reunión** Se hablará sobre el desarrollo de los puntos de control (checkpoints), además de revisar en conjunto el validador desarrollado.	--

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Desarrollo del validador de la aplicación, y revisión del asimilador	17/04/25	Iván Royo Gutiérrez
002	Desarrollar los scripts de compilación	24/04/25	Iván Royo Gutiérrez
003	Inicio de la parte de Implementación en la memoria	24/04/25	Iván Royo Gutiérrez
004	Desplegar aplicación DENIS Fiber en el entorno real	30/04/25	Jesús Carro Fernández

Reunión 7

REUNION:	**Revisión de Scripts y Validador, y Problemas de Firma en Linux ARM**

Fecha:	11/04/2025		
Hora comienzo:	9:45	**Hora finalización:**	10:45
Lugar:	Universidad San Jorge		
Elabora acta:	Iván Royo Gutiérrez		
Convocados:	Iván Royo Gutiérrez, Jesús Carro Fernández		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Disculpas por ausencia** No procede.	--
2	**Aprobar última acta** Se aprueba el acta de la reunión del 8 de abril de 2025.	--
3	**Asuntos pendientes última acta** No procede.	--
4	**Revisión del validador desarrollado** Jesús Carro Fernández, tras la ejecución de la primera prueba de control reducida a voluntarios, comenta que el validador está haciendo colas muy largas. Por lo que se compromete a revisar el mismo, además de revisar también el servidor, porque cree que puede ser un derivado de las nuevas medidas de seguridad aplicadas a este.	001
5	**Problemas de firma del binario de Linux ARM** Tras la primera versión lanzada a los voluntarios, se detectó un problema en la firma del binario del simulador para arquitecturas de Linux en ARM. Jesús Carro Fernández comento que había que crear un nuevo tag de versión sin cambios, pero necesario para distinguir entre versiones, ya que había que desplegar una nueva.	002 003
6	**Revisión Scripts de Compilación** Iván Royo Gutiérrez comento que resulta imposible compilar imágenes Windows desde su ordenador personal MacOS, aun probando un método encontrado con Vagrant, que no dio resultados. Por lo que se decidió separar la compilación de Windows a un entorno Windows, y la del resto de plataformas a un entorno MacOS.	--
7	**Otros asuntos** Quedan pendientes de comentar los puntos de control para la siguiente reunión, puesto que en esta se trataron asuntos más relevantes. Jesús Carro Fernández comento que mientras el probaba el validador, se siguiera avanzando en la memoria.	--
8	**Próxima reunión** Se discutirá acerca del desarrollo de los puntos de control, así como los problemas que estos conllevan para su prueba.	--

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Revisión del validador	30/04/25	Jesús Carro Fernández
002	Creación de la nueva versión en el repositorio	12/04/25	Iván Royo Gutiérrez
003	Despliegue de la nueva versión en la plataforma	14/04/25	Jesús Carro Fernández

Reunión 8

REUNIÓN:	**Puntos de Control y Problemas en compilados Windows**		
Fecha:	24/04/2025		
Hora comienzo:	9:45	**Hora finalización:**	10:45
Lugar:	Universidad San Jorge		
Elabora acta:	Iván Royo Gutiérrez		
Convocados:	Iván Royo Gutiérrez, Jesús Carro Fernández		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Disculpas por ausencia** No procede.	--
2	**Aprobar última acta** Se aprueba el acta de la reunión del 11 de abril de 2025.	--
3	**Asuntos pendientes última acta** Comentar los puntos de control (checkpoints) de la aplicación.	--
4	**Implementación de los Puntos de Control** Jesús Carro Fernández comenta que podría ser interesante llegar a tener una implementación de los checkpoints funcionando, ya que es algo que ofrece BOINC, pero requiere de una implementación propia. Este, comenta que serían relevantes para lanzar simulaciones largas, ya que así permitirá que estas guarden el estado ante imprevistos en los ordenadores voluntarios, o bien rotación de tareas. Para ello, Jesús comenta que hay unos existentes para la versión 0D, pero que requieren de adaptación a la nueva estructura.	001
5	**Problema de Optimización en binarios Windows** Iván Royo Gutiérrez comenta con Jesús Carro Fernández un problema potencial descubierto en el foro durante esta semana anterior de vacaciones. Iván Royo Gutiérrez comenta que lo ha conseguido solucionar, aplicando una migración de la compilación Windows a las VS Build Tools. Jesús Carro Fernández se compromete a lanzar la nueva versión que corrige este error, además de lanzar simulaciones de prueba para ver si esta mitigado.	002 003
7	**Otros asuntos** Jesús Carro Fernández comenta que se siga con el apartado de Implementación de la memoria, para así poder discutir en la siguiente reunión sobre los apartados posteriores.	004
8	**Próxima reunión** Se hablará sobre cómo ha ido la prueba de la nueva versión que arregla el problema de Windows, además de planear una posible nueva versión con puntos de control. También se comentará el inicio del siguiente apartado de la memoria, el Estudio Económico.	--

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Desarrollo de puntos de control (checkpoints).	17/05/25	Iván Royo Gutiérrez
002	Lanzamiento de la nueva versión a la plataforma	28/04/25	Jesús Carro Fernández
003	Lanzamiento de un nuevo lote de simulaciones a la red	29/04/25	Jesús Carro Fernández
004	Finalizar el apartado de Implementación de la memoria	17/05/25	Iván Royo Gutiérrez

Reunión 9

REUNIÓN:	**Despliegue de versión con Checkpoints y Revisión de la Memoria**

Fecha:	13/05/2025		
Hora comienzo:	9:45	**Hora finalización:**	10:45
Lugar:	Universidad San Jorge		
Elabora acta:	Iván Royo Gutiérrez		
Convocados:	Iván Royo Gutiérrez, Jesús Carro Fernández		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Disculpas por ausencia** No procede.	--
2	**Aprobar última acta** Se aprueba el acta de la reunión del 24 de abril de 2025.	--
3	**Asuntos pendientes última acta** No procede.	--
4	**Revisión de la memoria** Jesús Carro Fernández comenta que hará una revisión del apartado de la implementación de la memoria. Además comenta que sería relevante iniciar ya con el siguiente apartado, el del Estudio Económico, para ello, hizo especial énfasis en los gastos de la Seguridad Social para la empresa de los trabajadores.	001 002
5	**Despliegue de una Nueva Versión con Puntos de Control (Checkpoints)** Iván Royo Gutiérrez comenta que se ha terminado con el desarrollo de la nueva versión con los puntos de control integrados, además de añadir la posibilidad de modificar variables de los modelos globalmente. Jesús Carro Fernández comenta que se desplegará la nueva versión en la plataforma, además de lanzar un lote grande de tareas para testear la eficacia de estos, para así poder detectar posibles fallos.	003 004
6	**Otros asuntos** No se trataron otros asuntos importantes.	--
7	**Próxima reunión** Se hablará sobre las pruebas ejecutadas para la prueba de los puntos de control, y se empezaran a plantear ya las pruebas a realizar para el apartado de Resultados y Evaluación.	--

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Revisión de la memoria del proyecto	27/05/25	Jesús Carro Fernández
002	Realización del apartado de Estudio Económico	27/05/25	Iván Royo Gutiérrez
003	Despliegue de la nueva versión con puntos de control	20/05/25	Jesús Carro Fernández
004	Despliegue de un nuevo lote de tareas para la prueba de los puntos de control	27/05/25	Jesús Carro Fernández

Reunión 10

REUNIÓN:	**Análisis de Resultados Checkpoints y Pruebas para el Simulador**

Fecha:	27/05/2025		
Hora comienzo:	9:45	**Hora finalización:**	10:45
Lugar:	Universidad San Jorge		
Elabora acta:	Iván Royo Gutiérrez		
Convocados:	Iván Royo Gutiérrez, Jesús Carro Fernández		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Disculpas por ausencia** No procede.	--
2	**Aprobar última acta** Se aprueba el acta de la reunión del 13 de mayo de 2025.	--
3	**Asuntos pendientes última acta** No procede.	--
4	**Revisión de los resultados del lote para probar los Puntos de Control** Jesús Carro Fernández comenta que los resultados han ido bastante acordes a lo esperado, ya que ha habido tan solo 9 fallos, y todos repartidos en dos clientes, lo cual significa que han sido debido a problemas del cliente, y no de la lógica de los checkpoints. A pesar de esto, sugiere lanzar una nueva versión que modifique el output al archivo stderr.txt, de tal forma que se pueda acortar y ver donde se ha fallado.	001
5	**Prueba de variación de una conductancia** Jesús Carro Fernández sugiere realizar una prueba en la que se varie en cada posición de la fibra un ±25% la conductancia de la corriente Ks, para ello, sugiere crear un script para que te genere automáticamente las configuraciones divididas en sub-lotes de 1.000, siendo un total de 10 sub-lotes y 10.000 simulaciones. De esta prueba se rescatarían los tiempos de duración de la ejecución del lote, y se mostrarían varias graficas que demuestren que ha habido variación.	002 003
6	**Prueba de comparación con el simulador ELVIRA** Jesús Carro Fernández sugiere que sería interesante comparar los resultados de DENIS con los de ELVIRA, para así comprobar que la salida que generan ambos tiene que ser prácticamente idéntica, y no idéntica ya que usan diferentes métodos de resolución numéricos. Para ello, en ambas se haría bajo una fibra de las mismas características.	004 005
7	**Prueba de eficacia de los Puntos de Control** Iván Royo Gutiérrez sugiere realizar una prueba en la que se vea la eficacia de los checkpoints desarrollados, identificando así posibles errores. Para ello, se aprovecharía un nuevo lote de simulaciones para comprobar esto.	006
8	**Otros asuntos** No se trataron otros asuntos importantes.	--
9	**Próxima reunión** Se hablará sobre el resultado de las pruebas, así como de una última revisión de la memoria antes de la entrega final.	--

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Desplegar nueva versión que acorte el output a stderr.txt	30/05/25	Iván Royo Gutiérrez
002	Crear script que genere las configuraciones para la prueba	03/06/25	Iván Royo Gutiérrez
003	Lanzar las simulaciones con variación de la conductancia	10/06/25	Jesús Carro Fernández
004	Preparar y ejecutar una simulación base de DENIS	10/06/25	Iván Royo Gutiérrez
005	Preparar y ejecutar una simulación idéntica a DENIS en ELVIRA.	10/06/25	Jesús Carro Fernández
006	Mandar un lote de simulaciones para probar los Puntos de Control bajo la nueva versión	03/06/25	Jesús Carro Fernández

Reunión 11

REUNIÓN:	**Lanzamiento de una versión definitiva de DENIS Fiber**

Fecha:	03/06/2025		
Hora comienzo:	9:45	**Hora finalización:**	10:45
Lugar:	Universidad San Jorge		
Elabora acta:	Iván Royo Gutiérrez		
Convocados:	Iván Royo Gutiérrez, Jesús Carro Fernández		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Disculpas por ausencia** No procede.	--
2	**Aprobar última acta** No procede.	--
3	**Asuntos pendientes última acta** No procede.	--
4	**Lanzamiento de versión definitiva** Ante los buenos resultados que están dando los checkpoints tras la ejecución del nuevo lote, Jesús Carro Fernández comenta que vamos a lanzar una nueva versión definitiva del simulador. También comenta que use el lote de tareas que se han completado para la prueba de los puntos de control en el apartado de Resultados y Evaluación.	001 002
5	**Realización del resto de pruebas bajo la versión definitiva** Jesús Carro Fernández comenta que las pruebas restantes se pueden hacer bajo la versión definitiva en la plataforma, puesto que así un mayor número de voluntarios se suele sumar con respecto a aplicaciones en beta.	--
6	**Otros asuntos** Se mantienen los plazos indicados en el acta anterior para las pruebas restantes.	--
7	**Próxima reunión** Se comentarán los resultados de las pruebas restantes y también se hablará de una revisión final de la memoria del trabajo de cara a la entrega.	--

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Creación de una nueva versión definitiva de la aplicación en el repositorio.	04/06/25	Iván Royo Gutiérrez
002	Despliegue de una nueva aplicación dentro del proyecto de BOINC, pero esta vez sin marcar que es beta.	10/06/25	Jesús Carro Fernández, Iván Royo Gutiérrez

Reunión 12

REUNIÓN:	**Análisis de Pruebas Obtenidas, Realización de la Memoria y Revisión**

Fecha:	11/06/2025		
Hora comienzo:	9:45	**Hora finalización:**	10:45
Lugar:	Universidad San Jorge		
Elabora acta:	Iván Royo Gutiérrez		
Convocados:	Iván Royo Gutiérrez, Jesús Carro Fernández		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Disculpas por ausencia** No procede.	--
2	**Aprobar última acta** Se aprueba el acta de la reunión del 3 de junio de 2025.	--
3	**Asuntos pendientes última acta** No procede.	--
4	**Scripts para el análisis de los resultados** Jesús Carro Fernández e Iván Royo Gutiérrez hablan sobre crear unos scripts que desarrollen unas graficas con los datos obtenidos de las pruebas de DENIS frente a ELVIRA, para la comparación del potencial de acción (AP) y los APD relevantes, así como para la variación de la conductancia unos que muestren el progreso de las tareas en el tiempo, la variación de los AP frente a uno base, y los diferentes APD en un histograma extraídos del archivo de postprocesado.	001 002
5	**Realización de los apartados restantes de la memoria** Jesús Carro Fernández comenta plasmar con estas gráficas los resultados de las pruebas obtenidas en la memoria, así como realizar ya los apartados restantes de la misma.	003
6	**Revisión final de la memoria** Jesús Carro Fernández se compromete a hacer una última revisión de la memoria, leyéndose la estructura entera del trabajo para así analizarlo entero en búsqueda de posibles erratas u errores gramaticales.	004
7	**Otros asuntos** No se trataron otros asuntos importantes.	--
8	**Próxima reunión** De haberla, se comentaran las correcciones de la memoria, así como la interpretación de las gráficas obtenidas.	--

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Desarrollo de scripts para mostrar una gráfica y marcadores de ambos AP de DENIS frente a ELVIRA.	13/06/25	Iván Royo Gutiérrez
002	Desarrollo de scripts para la prueba de la conductancia, que generen progreso del tiempo, APs e histogramas de APD de las simulaciones obtenidas de la prueba.	14/06/25	Iván Royo Gutiérrez
003	Redacción de los resultados obtenidos de las pruebas, así como de las conclusiones en la memoria.	16/06/25	Iván Royo Gutiérrez
004	Revisión para la entrega final de la memoria del proyecto.	19/06/25	Jesús Carro Fernández

Anexo III – Autorización de uso de herramientas de IA

D. Jesús Carro Fernández con DNI 71664373F, tutor del proyecto de título "Computación voluntaria para la simulación de tejidos cardiacos heterogéneos" por el alumno Iván Royo Gutiérrez con DNI 79353485N, **AUTORIZO** expresamente al uso de herramientas de Inteligencia Artificial (IA) en el contexto de dicho TFG con el siguiente ámbito y fines de uso:

- Ámbito: Uso de herramientas de Inteligencia Artificial en tareas de apoyo, análisis y generación de recursos relacionados con la implementación de un nuevo simulador en la plataforma DENIS para la simulación multiescala de tejidos cardiacos heterogéneos. Estas herramientas se emplearán exclusivamente como apoyo en tareas de revisión de la documentación elaborada y de apoyo en la comprensión del conocimiento previo del proyecto en el que se enmarca este TFG, garantizando en todo momento el cumplimiento de los principios éticos y académicos establecidos por la Institución.
- Finalidad de uso 1: Asistir en la revisión y comprensión del simulador unicelular original, facilitando su comprensión y respondiendo a preguntas sobre su estructura para así facilitar que se pueda mantener la coherencia con el mismo en el nuevo simulador multiescala.
- Finalidad de uso 2: Apoyar en la revisión de la memoria del TFG, de cara a la mejora ortográfica y gramatical. La IA se empleará también para revisar la coherencia y claridad del documento, ayudando en la búsqueda de fragmentos en los que sea difícil la comprensión o el lenguaje sea muy repetitivo.

En Zaragoza, a 20 de noviembre de 2024



Fdo: Jesús Carro Fernández

Anexo IV – Problema de optimización en compilados Windows

Durante la fase de pruebas de la versión beta 0.02 del proyecto DENIS Fiber, surgieron reportes significativos de los usuarios voluntarios sobre una notable pérdida de rendimiento en las simulaciones ejecutadas en sistemas operativos Windows, en comparación con otras plataformas como Linux y macOS. Estos reportes fueron centralizados en un hilo específico del foro oficial del proyecto, donde los voluntarios solicitaron una revisión técnica profunda para resolver dicho inconveniente (https://denis.usj.es/denisathome/forum_thread.php?id=314).

Tras realizar un análisis, se determinó que la causa principal del problema era debido al uso del compilador MinGW para generar los binarios en Windows. Aunque MinGW ofrece compatibilidad y facilidad inicial en la generación de binarios, carecía de capacidades avanzadas de optimización específicas para Windows, generando un ejecutable que, aunque funcional, era considerablemente menos eficiente.

Con el objetivo de solventar este problema, en la versión beta 0.03 se decidió migrar completamente hacia Visual Studio Build Tools, la herramienta oficial y especializada proporcionada por Microsoft para la compilación y optimización de código en entornos Windows. Este cambio permitió generar binarios mucho más optimizados (ya que al compilar, genera directamente instrucciones específicas para Windows), resultando en una mejora significativa del rendimiento del simulador en dicho sistema operativo, llegando incluso a ser de hasta cuatro veces mayor con respecto a la versión anterior compilada con MinGW.

Aunque dicho cambio no fue tan simple como parece. Esto se debe a que BOINC, en general, asume que vas a tratar de compilar las librerías desde una interfaz gráfica, la cual en el momento de la instalación te instala todas las dependencias necesarias para poder usar las VS Build Tools sin problemas mayores. Por lo que debido a esto, tratar de compilarlo por consola supuso un verdadero reto, ya que no te dicen exactamente cuáles son los componentes que necesitas para compilar exactamente estas herramientas en la documentación.

Es por eso por lo que se tuvo que ir buscando a base de prueba y error que dependencias eran las que necesitaba la librería de BOINC para compilarse con éxito, haciendo de este proceso algo tedioso y muy prolongado en el tiempo, ya que cada vez que daba error al compilarlo, se debía de editar el Dockerfile para así añadir más dependencias, basándonos en todo momento en las que tenía en mi sistema Windows local (las ya conocidas). A continuación se muestran las dependencias que requiere BOINC para poder ser compilado completamente desde consola:

```
RUN `
    curl -SL --output vs_buildtools.exe https://aka.ms/vs/17/release/vs_buildtools.exe && `
    (start /w vs_buildtools.exe --quiet --wait --norestart `
        --installPath "%ProgramFiles(x86)%\Microsoft Visual Studio\2022\BuildTools" `
        --add Microsoft.VisualStudio.Workload.VCTools `
        --add Microsoft.VisualStudio.Component.VC.Tools.x86.x64 `
        --add Microsoft.VisualStudio.Component.Windows10SDK.20348 `
        --add Microsoft.VisualStudio.Component.VC.CMake.Project `
        --add Microsoft.VisualStudio.Component.TextTemplating `
        --add Microsoft.VisualStudio.Component.VC.CoreBuildTools `
        --add Microsoft.VisualStudio.ComponentGroup.NativeDesktop.Core `
        --add Microsoft.VisualStudio.Component.VC.Redist.14.Latest `
        --add Microsoft.VisualStudio.Component.VC.ATL `
        --locale en-US `
    || IF "%ERRORLEVEL%"=="3010" EXIT 0) && `
    del /q vs_buildtools.exe
```

Figura 29: Componentes necesarios para la compilación BOINC con VS Build Tools.

En la imagen, podemos ver todos estos componentes necesarios para la correcta compilación de las librerías requeridas por el simulador de BOINC, para así solucionar este problema de optimización.

Ahora bien, aunque el binario es hasta cuatro veces más rápido, la construcción de la imagen de Docker es en esta nueva versión mucho más lenta:

- En la versión beta_0.02, correspondiente a la compilación con MinGW, de media la imagen de Docker en el equipo habitual de trabajo Windows tardaba en construirse alrededor de 5-6 minutos.
- En la versión beta_0.03, correspondiente a la compilación con las VS Build Tools, de media la imagen de Docker tarda unos 35-40 minutos en construirse en el mismo equipo.

Por lo que supone una construcción de la imagen mucho más lenta para la nueva versión que para la anterior.

Aunque, en términos de optimización, es mucho mejor la nueva versión. Esto se debe a que una imagen de Docker, una vez la construyes ya no hace falta que lo vuelvas a hacer más, simplemente tienes que desplegarla (mediante el comando "*$docker run …*") llamando al compilador que se crea en su interior, para que así esta te genere el binario compilado (que posteriormente te traes a tu máquina). En términos de compilación, ambas versiones de la imagen de Docker tardan más o menos lo mismo en compilar el simulador, siendo este tiempo alrededor de unos 2 minutos.