Trabajo Fin De Grado:

El requisito de sostenibilidad en una aplicación de medición de impacto

Grado en Ingeniería Informática
Ingeniería de Computadores

27 de abril de 2022

Autor: David Rodriguez Redondo
Directora: Marisa Gil Gomez

Resumen

El propósito de este trabajo es responder a la pregunta sobre la viabilidad de ejecutar una aplicación, cuyas tareas principales son la comunicación con una base de datos, en diferentes entornos.
Tradicionalmente, la ejecución de un programa viene gestionada por una Unidad de Procesamiento Central (CPU). Sin embargo, se ha observado el gran potencial de las Unidades de Procesamiento Gráfico (GPU) para ejecutar tareas de mayor complejidad o que requieren un procesado de datos muy elevado. Con este proyecto quieren observarse las implicaciones que supone ejecutar la misma aplicación en un escenario, donde la mayor parte de tareas las ejecuta una GPU.
Finalmente, se tendrán en cuenta las implicaciones energéticas y de rendimiento para determinar si esta transición es realmente una opción viable.

Resum

El propòsit d'aquest projecte és poder respondre a la pregunta sobre la viabilitat d'executar una aplicació, la qual es comunicarà amb una base de dades com a tasca principal, en diferents entorns.
Tradicionalment, l'execució d'un programa ha estat gestionada per la Unitat de Processament Central (CPU). Tanmateix, s'ha pogut observar el gran potencial de les Unitats de Processament Gràfic (GPU) per executar tasques de major complexitat o que requereixen un processament de dades més elevat. En aquesta memòria s'analitzaran les implicacions que suposa executar la mateixa aplicació en un escenari on la major part de les funcions seran executades per una GPU.
Finalment, es tindrà en compte les implicacions energètiques i de rendiment per determinar si aquesta transició és realment una opció viable.

Abstract

The aim of this project is to answer the question on the viability of running an application, of which the main tasks consist in communicating with a database, in different environments. Traditionally, the Central Processing Unit (CPU) oversees the execution of a program. Nonetheless, Graphic Processing Units (GPU) have shown a high potential when processing big amounts of data or more complex information. The goal is to observe the different implications that migrating from one environment to another, will have.
All in all, energy usage and performance implications will be considered, to determine whether this alternative is a viable solution, or not.

Índice

Listado de tablas

Listado de figuras

1. Introducción

La sostenibilidad es un criterio importante y decisivo para todo aquel trabajo desarrollado por empresas, definiendo así la cultura de esta y el impacto en la sociedad.
El objetivo de esta memoria es evaluar de forma sostenible el entorno para acoger a una aplicación que se comunica con una base de datos. Se plantean dos escenarios diferentes, el primero será hacer correr la aplicación en una máquina Intel, donde la CPU se encarga de la mayor parte de la computación. Por otro lado, se migrará la misma aplicación a una máquina donde la GPU haga la mayor parte de los cálculos computacionales. La meta es adaptar las operaciones a este segundo modelo. Posteriormente, se analizarán las diferencias entre estos.

1.1. Contextualización

La información con la que se trabajará en la base de datos ha sido extraída por los compañeros David Román Camps y Carles Pàmies Montero, durante el desarrollo de una aplicación, que marca como objetivo facilitar la búsqueda de información relacionada con una ONG, para permitir a los usuarios hacer decisiones sobre estas (colaboraciones, información general…), al igual que una función para filtrar aquellas ONG con las que finalmente se trabajará.

Durante el proyecto se llenará una BBDD con la información recogida, para poder desarrollar las diferentes pruebas, que, en su conjunto, determinarán que entorno es el más eficiente para computar las operaciones de una base de datos. Se quiere llevar a cabo un análisis de rendimiento, evaluar el gasto energético y determinar la plataforma hardware donde se ejecutará la aplicación, es decir, todos aquellos parámetros (base de datos, arquitecturas…) que contribuyan a la mejor solución.

Este Trabajo de Fin de Grado (TFG) pertenece al *Grado de Ingeniería Informática* en la especialización de Ingeniería de Computadores impartido por la Universidad Politécnica de Cataluña. Además, se clasifica en Modalidad A.

1.2. Definición de términos claves

En este apartado se definirán todos aquellos conceptos que puedan causar confusión, o bien, facilitar la comprensión a lo largo del proyecto.

<u>Bases de datos</u>
Programa capaz de almacenar gran cantidad de datos, que formen parte de un mismo contexto. Estos pueden ser consultados rápidamente en función de las características selectivas que se deseen.

Arquitectura de un procesador

Se denomina arquitectura a la estructura interna de un procesador. La distribución y ubicación de las unidades lógicas y físicas (ALU, Registros, Unidad de Control…). Existen dos tipos principales y estos son RISC y CISC con sus respectivas ventajas y desventajas. Las diferencias entre estas se estudiarán más adelante.

1.3. Identificación del problema

El reto con el que se trabaja en la memoria, es la viabilidad de ejecución de una base de datos en un entorno que derive la mayor parte de los cálculos a una Unidad de Procesamiento Gráfico, en vez de delegarlo a la CPU. El objetivo es encontrar el entorno más eficiente para ejecutar la aplicación.

Esta cuestión nace a raíz de la evolución de las GPU durante los últimos años, ya que el potencial que ofrecen les podría permitir competir directamente con las CPU, en vez de limitarse a ser un complemento o ayuda para estas.

Durante este proyecto se evaluará el rendimiento de diferentes operaciones ejecutadas por una base de datos, para poder determinar, a partir de las ventajas y desventajas, que opciones pueden ser las más beneficiosas para este caso. Por lo tanto, se espera que al final de la memoria se recojan suficientes datos para poder determinar que entorno es más eficiente para ejecutar una aplicación que hace consultas a una base de datos.

1.4. Actores implicados

Los actores implicados en un proyecto, también conocidos como *stakeholders*, son aquellas figuras clave para el correcto desarrollo de este. En este apartado se definen los actores tanto beneficiarios, como los participantes en el desarrollo.

El principal beneficiario serán los desarrolladores de la aplicación, ya que gracias a la información proporcionada en esta memoria serán capaces de crear un entorno sostenible y utilizar el entorno más adecuado para el desarrollo del proyecto.

2. Justificación

2.1. Análisis de mercado

Podemos encontrar una gran variedad de tipos de Bases de Datos (BBDD), con sus ventajas y desventajas correspondientes, y por eso resulta necesario encontrar aquella que mejor se adapte a este proyecto. La gran diferencia entre estas se rige por sí son una Base de Datos SQL o NoSQL.

El primer tipo permite combinar diferentes tablas de forma muy eficiente, con el objetivo de extraer información relacionada. Mientras que NoSQL lo hace de forma muy limitada o directamente no lo permite. Asimismo, con una BBDD SQL se pueden gestionar los datos en conjunto con las relaciones que existen entre estos y, sin embargo, en NoSQL este tipo de gestión tampoco se permite.
Finalmente, NoSQL permite que se gestionen grandes cantidades de información, mientras que SQL permite distribuir bases de datos relacionales.

Tabla 1: Diferencias entre SQL vs NoSQL. Fuente: Elaboración propia

SQL	NoSQL
Relacional	No relacional
Estructura en tablas	Estructura en documentos JSON, Key/value …
Esquema estricto	Esquema flexible
Lenguaje SQL	Otros

Por otro lado, se comparará el funcionamiento de una CPU y una GPU. La primera es un procesador muy potente, que dirige al resto de componentes de un ordenador, así como haría el cerebro en un cuerpo humano. Por otro lado, la GPU suele utilizarse como un componente que acompaña a la CPU, y su objetivo es ejecutar aquellas operaciones más complejas o que requieren de más recursos. Además, la CPU suele trabajar de forma lineal y la GPU acostumbra a paralelizar la mayor parte de los procesos.

2.2. Conclusión

La aplicación inicialmente fue desarrollada en la base de datos Redis. Sin embargo, para esta memoria se hace un estudio de las diferentes opciones, así como el funcionamiento de cada una de estas. Con la información recogida, se confirma si utilizar Redis es una buena opción para el entorno GPU, o si, en caso contrario, existe una mejor alternativa.

3. Alcance

3.1. Objetivos

Con la memoria se quiere evaluar el rendimiento de la base de datos en diferentes arquitecturas. Es decir, estudiar las operaciones que hace, como selección de, inserción o eliminación de datos...
Además, para llevar a cabo la evaluación del rendimiento, deben estudiarse las distintas arquitecturas o entorno hardware en cuanto a la utilización de BBDD. Es decir, como los procesadores se comunican con la memoria.

Para poder determinar de modo óptimo el mejor entorno hardware para el desarrollo de la aplicación es necesario cumplir los siguientes objetivos. Además, se listan también los requisitos y riesgos que pueden aparecer a lo largo de la implementación.

- **Determinación de la arquitectura hardware para el entorno de desarrollo**. Hacer un estudio de diferentes entornos y su arquitectura. Además, estudiar como estos se gestionan la memoria y otros componentes.

- **Estudio del entorno más eficiente**. Determinar que parámetros pueden determinar que entorno es el más sostenible y eficiente para acoger a una aplicación que hará consultas a una base de datos.

3.2. Requisitos no funcionales

Los requisitos no funcionales son aquellos no son necesarios para el funcionamiento, pero sí proporcionan un mejor resultado, ya que contribuyen a una mejor experiencia para el usuario.

- **Eficiencia**. El entorno donde se desarrollará la aplicación debe ser eficaz, es decir, proporcionar los resultados esperados sin generar un consumo de recursos excesivo.

- **Escalabilidad**. Idealmente, la plataforma hardware debe ser escalable. Para que en caso de que las necesidades se vean incrementadas, se pueda readaptar únicamente incrementando determinadas capacidades de la plataforma hardware.

3.3. Riesgos

Cuando se desarrolla un proyecto es necesario hacer una evaluación de los posibles riesgos que pueden surgir durante este. Nuestro objetivo es conocerlos e intentar minimizar el impacto.

- **Deterioro de la plataforma hardware.** Todo componente electrónico tiene una vida útil limitada y es necesario conocer esta duración, ya que una renovación de estos componentes demasiado frecuentes podría afectar el gasto económico, así como incrementar el impacto ambiental del proyecto.

- **Falta de integración o compatibilidad**. En caso de que los desarrolladores de la aplicación decidan hacer un cambio radical en la base de datos o en el rendimiento del programa, podría verse afectada el entorno que se describe en el trabajo, y provocar una falta de compatibilidad entre estos.

4. Metodología

4.1. Metodología de trabajo

Determinar la metodología para un proyecto es crucial, ya que determina el ritmo de desarrollo y la forma de trabajo. Además, es muy importante a la hora de enfrentar problemas.

Para desarrollar este trabajo se utilizará la metodología *Agile*, que permitirá adaptarse a los cambios que puedan tomarse a lo largo del proyecto y, a su misma vez, revisar continuamente la evolución de este.

4.2. Herramientas de desarrollo y seguimiento

Las herramientas de desarrollo o seguimiento que se emplearán para llevar a cabo el trabajo son las siguientes.

La información obtenida para llevar a cabo la investigación se hará principalmente mediante recursos online, como artículos, páginas web... de donde se extrae, en mayor parte, la información que se recoge en el trabajo.

Por otro lado, inicialmente se estudió el uso del Clúster del *Computer Architecture Departmen*t (DAC), en cuanto a funcionamiento del entorno y comunicación con este. Sin embargo, finalmente se empleó la herramienta Gradient la cual permite hacer pruebas en GPU mediante la creación de Notebooks. Todo este proceso y tecnicismos son explicados a lo largo de la memoria.

Como método de seguimiento se establecieron dos reuniones con la tutora del TFG, Marisa Gil, que tuvieron lugar en el inicio del proyecto. El objetivo de estas era conseguir una mayor familiarización con el proyecto.

5. Planificación temporal

Inicialmente, se estimó una duración de 4 meses para el desarrollo del trabajo. Sin embargo, debido a inconvenientes personales, el trabajo ha tenido que ser aplazado unos meses más, lo cual ha implicado que varias tareas hayan sido modificadas, sobre todo aquellas que están relacionadas con el desarrollo e investigación del proyecto. Finalmente, la fecha de inicio se define el día 20 de septiembre de 2021 y la fecha final para llevar a cabo la presentación tiene lugar el día 27 de abril de 2022

Para una mejor comprensión de estos cambios, a continuación, se expondrán las diferencias entre ambas planificaciones, donde se verán los cambios reflejados causados por el aplazamiento. Finalmente, se presentará el nuevo diagrama de Gantt, así como los cambios en el presupuesto.

En total, se calcula una duración de 488,25 horas para poder finalizar por completo el proyecto.

5.1. Fases del proyecto y recursos

Para llevar a cabo el proyecto se ha dividido el trabajo en 4 fases principales y una fase final. Se trabajará mediante *Sprints*, un periodo de tiempo corto en el que se trabaja de forma intensiva. Sin embargo, inicialmente las tareas que se describieron en cada fase no concuerdan con la realidad, ya que finalmente se abordó el proyecto de un modo diferente. Para poder comprender de manera más sencilla las diferencias entre las dos planificaciones no se modificará el ID que se asigna a cada tarea.
El orden natural de estas era seguir un orden creciente en los números, por ejemplo: S1, S2, S3, R1, S4…. Ahora, el orden es S1, S6, S7, R1, S2…

Con relación a los recursos humanos necesarios, encontramos las figuras que forman parte en el desarrollo del proyecto. En este caso la principal persona al cargo de la memoria soy yo, David Rodríguez. Además, cuento con la ayuda de mi tutora del TFG, así como con una mentora que se ha encargado de guiarme durante la parte inicial relacionada con la gestión del proyecto.

Sin embargo, al considerar los recursos software y hardware, se observa que estos son recurrentes durante las diferentes etapas. A continuación, se listarán.

Un ordenador para poder llevar a cabo la memoria, así como la obtención de información. Consecuentemente, será necesario una conexión a internet. Además, el acceso a la plataforma DAC para poder hacer mediciones y comparaciones de rendimiento.

5.2. Definición tareas de cada fase

Gestión del proyecto

- G1 - Contextualización y Alcance
  Resumen: Definición del alcance del proyecto en su contexto. Especificación general, contextualización y justificación.
  Duración: 24,50 h (20/9/21 - 28/9/21)
  Dependencias: -

- G2 - Planificación temporal
  Resumen: Descripción de las diferentes fases del proyecto y recursos correspondientes para cada una.
  Duración: 8,25 h (29/9/2021 - 4/10/21)
  Dependencias: G1

- G3 - Gestión económica y sostenibilidad
  Resumen: Autoevaluación sobre la sostenibilidad, estudio de la dimensión económica y presupuesto.
  Duración: 9,25 h (5/10/21 - 11/10/21)
  Dependencias: G2

- G4 - Documentación final
  Resumen: Aplicar cambios y mejoras a las tareas previas. Finalmente, crear el documento final donde se recoja toda la información.
  Duración: 18,25 h (12/10/21 - 18/10/21)
  Dependencias: G3

Sprint 1

- S1 - Inicio del proyecto
  Resumen: Iniciar investigación sobre diferentes plataformas hardware.
  Duración: 40 h (19/10/21 – 25/12/21)
  Dependencias: -

- S6 - Estudio coste energético de la plataforma hardware
  Resumen: Investigar sobre el impacto ambiental del entorno elegido.
  Duración: 70 h (26/12/21 - 31/1/22)
  Dependencias: S1

- S7 - Comparación con otras plataformas
  Resumen: Redactar una comparación entre las diferentes alternativas y hacer un estudio sobre ventajas y desventajas.
  Duración: 80 h (1/2/22 - 28/2/22)
  Dependencias: S6

- R1 - Documentación final
  Resumen: Recoger toda la información redactada y aplicar los cambios necesarios.
  Duración: 25 h (1/3/22 - 25/3/22)
  Dependencias: S7

Sprint 2

- S2 - Determinar plataforma hardware
  Resumen: Decisión y discusión con desarrolladores para determinar la mejor plataforma hardware.
  Duración: 30 h (26/3/22 - 29/3/22)
  Dependencias: R1

- S3 - Decisión arquitectura hardware
  Resumen: Hacer un estudio sobre diferentes arquitecturas hardware y contrarrestar los datos con la decisión previamente elegida en S2.
  Duración: 30 h (30/3/22 - 2/4/22)
  Dependencias: S2

- R2 - Documentación final
  Resumen: Redactar un documento final que recoja toda la información previa y aplique las correcciones correspondientes.
  Duración: 15 h (3/4/22 - 4/4/22)
  Dependencias: R1, S3

Sprint 3

- S4 - Familiarización del entorno DAC / Gradient
  Resumen: Familiarización con el entorno DAC y Gradient para entender el funcionamiento y prestaciones de este.
  Duración: 30 h (5/4/22 - 8/4/22)
  Dependencias: -

- S5 - Evaluación del rendimiento en plataforma Gradient
  Resumen: Evaluar el rendimiento en ambos entornos hardware
  Duración: 30 h (9/4/22 - 17/4/22)
  Dependencias: S4

- R3 - Documentación final
  Resumen: Redactar un documento final donde se recojan los datos obtenidos a través de Gradient y aplicar las revisiones correspondientes.
  Duración: 16 h (18/4/22 - 20/4/22)
  Dependencias: R2, R3

Fase Final

- F1 - Memoria final
  Resumen: Redactar la memoria final.
  Duración: 30 h (21/4/22 - 23/4/22)
  Dependencias: R3

- F2 - Preparación defensa del TFG
  Resumen: Preparación de la presentación de la memoria.
  Duración: 30 h (20/4/22 - 24/2/22)
  Dependencias: F1

Tareas extras

- E1, E2 - Reuniones de seguimiento
  Resumen: Reuniones iniciales con la tutora del proyecto.
  Duración: 2 h
  Dependencias: -

5.3. Planificación inicial

En la siguiente tabla se pueden ver todas las tareas que se plantean inicialmente.

Tabla 2: Tabla estimaciones inicial. Fuente: Elaboración propia

ID	Tarea	Horas	Dependencias
G1	Contextualización y alcance	24,5	-
G2	Planificación temporal	8,25	G1
G3	Gestión económica y sostenibilidad	9,25	G2
G4	Documentación final	18,25	G3
	FASE 1		
S1	Inicio del proyecto	30	-
S2	Determinar plataforma hardware	25	S1
S3	Decisión arquitectura hardware	30	S2
R1	Documentación final	15	S3
	FASE 2		
S4	Familiarización del entorno DAC	50	-
S5	Evaluación del rendimiento en plataforma DAC	60	S4
R2	Documentación final	16	R1, S5
	FASE 3		
S6	Estudio coste energético de la plataforma hardware	25	-
S7	Comparación con otras plataformas	40	S6
R3	Documentación final	25	R2, S7
	FASE 4		
F1	Memoria final	50	R3
F2	Preparación defensa del TFG	30	F1
E1...9	Reuniones de seguimiento	9	-
TOTAL		**465,25**	

5.4. Planificación final

A continuación, en la tabla 2 se puede observar como las etapas marcadas en negrita han sufrido un cambio en la duración. Para una mejor comprensión, se ha mantenido la descripción de la tarea, pero se detallará el porqué de estos cambios.

Como se ha comentado previamente, debido a inconvenientes personales y de desarrollo, se ha tenido que aplazar la entrega final del trabajo. Esta decisión tiene un impacto relativamente bajo cuando se comparan el número de horas totales entre ambas planificaciones.

La estimación y organización inicial se mantienen. Es decir, se sigue trabajando mediante *Sprints*, los cuales han permitido una fácil adaptación a este aplazamiento, ya que se trabaja en tareas pequeñas de trabajo intensivo.

Para la Fase 1 se han necesitado más horas, puesto que el periodo de investigación y complejidad, eran superiores a los estimados. Por otro lado, al aumentar el tiempo dedicado en esta etapa, se estima que las horas dedicas al resto de fases sean menores, ya que se entrará en estas fases con mayor conocimiento, y no se requerirá tantas horas para familiarizarse con el entorno.

Finalmente, se ha determinado que para un mejor desarrollo del proyecto se reorganice el orden de los *Sprints*. El orden de las fases ha quedado modificado, y los *Sprints* que se encontraban en la fase 1,2 y 3 respectivamente, ahora se organizan como 3,1 y 2.
Por lo tanto, ahora en la primera fase se hace un estudio de las diferentes arquitecturas y se comparan. En la segunda fase se determinará y decidirá cuál es la arquitectura elegida. Y en la tercera fase se hará el estudio práctico. El resto de las fases no han sido modificadas.

Esta nueva organización ha aumentado en 23 horas el tiempo total que se necesitará para el proyecto. Como se comenta en el inicio, este aplazamiento ha tenido un impacto leve, ya que se ha pospuesto mayormente por falta de tiempo y no tanto por dificultad. Eso se traduce en un mayor tiempo de desarrollo, donde se extiende unos meses. Así pues, a continuación, se puede observar en la tabla 2 la nueva estimación que permitirá obtener los mismos resultados.

Tabla 3: Tabla estimaciones final. Fuente: Elaboración propia

ID	Tarea	Horas	Dependencias
G1	Contextualización y alcance	24,5	-
G2	Planificación temporal	8,25	G1
G3	Gestión económica y sostenibilidad	9,25	G2
G4	Documentación final	18,25	G3
	FASE 1		
S1	Inicio del proyecto	**40**	-
S6	**Estudio coste energético de las plataformas hardware**	**70**	S1
S7	**Comparación con otras plataformas**	**80**	S6
R1	**Documentación final**	**25**	S7
	FASE 2		
S2	**Determinar plataforma hardware**	**30**	R1
S3	**Decisión arquitectura hardware**	**30**	S2
R2	**Documentación final**	**15**	R1, S3
	FASE 3		
S4	**Familiarización del entorno DAC / Gradient**	**30**	-
S5	**Evaluación del rendimiento en plataforma Gradient**	**30**	S4
R3	**Documentación final**	**16**	R2, R3
	FASE 4		
F1	Memoria final	30	R3
F2	Preparación defensa del TFG	30	F1
E1, E2	Reuniones de seguimiento	2	-
TOTAL		**488,25**	

6. Diagrama Gantt

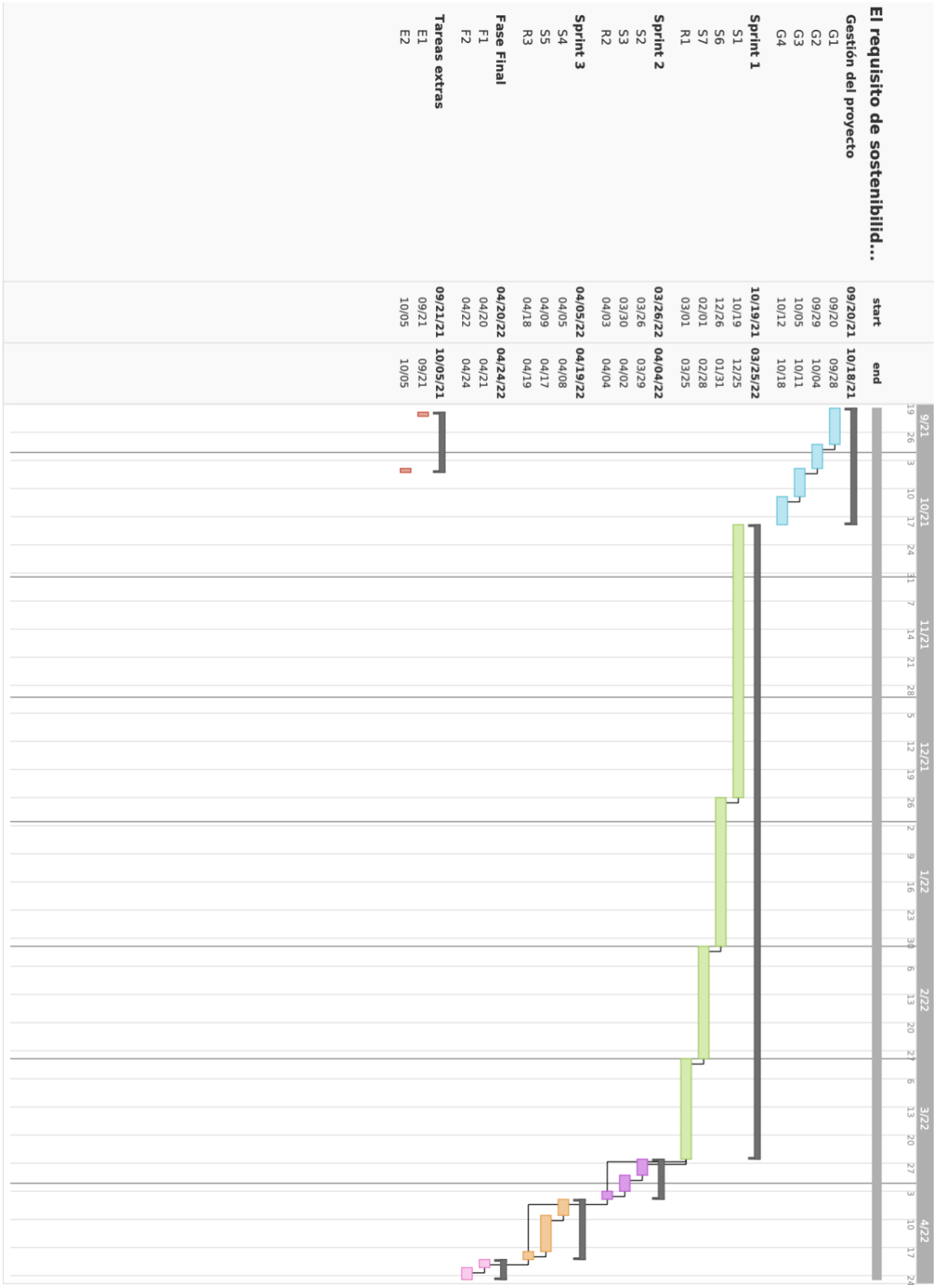

Figura 1: Diagrama de Gantt. Fuente: Elaboración propia

7. Gestión del riesgo: Planes alternativos y obstáculos

En este apartado se hace un planteamiento de los diferentes riesgos y obstáculos que pueden aparecer en el desarrollo del proyecto. Sin embargo, se listarán además posibles soluciones y alternativas para estos mismos inconvenientes.

- **Deterioro de la plataforma hardware.** Para poder reducir el deterioro que pueden tener determinados componentes hardware, la mejor estrategia es apostar por aquellos componentes con una larga vida útil, así como plantear estos gastos en el estudio económico para que el cliente pueda aceptarlos y prevenirlos.

- **Falta de integración o compatibilidad**. La forma de solucionar este problema es muy similar al anterior, y eso conlleva principalmente, una buena comunicación con los desarrolladores de la aplicación, para que cualquier cambio se vea reflejado en ambas partes y no se produzca ninguna incompatibilidad de último momento.

8. Presupuesto

8.1. Fases del proyecto y recursos

Para poder hacer una correcta estimación del coste del proyecto es necesario evaluar los gastos materiales, así como los recursos, software y hardware. Además, también hay que reflejar los costes humanos, es decir, las personas que han desarrollado el trabajo.

Debido al aplazamiento, el presupuesto inicial, que se fijaba en 12.371,27 €, también sufrió cambios. A continuación, se detallarán directamente los datos finales, haciendo énfasis en las cifras que han sufrido un cambio.

Recursos humanos

Los roles se han dividido para simular un equipo de trabajo, pudiendo asignar diferentes actividades a cada rol. Cabe destacar que la información sobre el sueldo asignado a cada uno ha sido extraída de la plataforma *Glassdoor*.

Tabla 4: Presupuesto recursos humanos. Fuente: Elaboración propia

Recursos humanos	Sueldo / hora (neto)	Sueldo / hora (bruto)
Jefe de proyectos	22,5 €	29,25 €
Desarrollador	18 €	23,4 €
Tester	15,6 €	20,28 €

A continuación, se calcula el coste total según las horas trabajadas por rol establecido.

Tabla 5: Coste recursos humanos. Fuente: Elaboración propia

ID Tarea	Horas	Jefe de proyectos (h)	Desarrollador (h)	Tester (h)	Total (€)
G1	24,5	24,5	-	-	716,63
G2	8,25	8,25	-	-	241,31
G3	9,25	9,25	-	-	270,56
G4	18,25	18,25	-	-	533,81
S1	**40**	**40**	-	-	1.170,00

S2	**30**	**5**	**25**	-	731,25
S3	**30**	**5**	**25**	-	731,25
R1	15	15	-	-	438,75
S4	**30**	-	**30**	-	702,00
S5	**30**	-	-	**30**	608,40
R2	16	16	-	-	468,00
S6	70	70	-	-	2.047,50
S7	80	70	-	10	2.250,30
R3	25	25	-	-	731,25
F1	30	30	-	-	877,50
F2	30	30	-	-	877,50
E1, E2	2	2	-	-	58,50
TOTAL	488,25	368,25	80	40	**13.454,51**

Por lo tanto, si comparamos el presupuesto inicial 12.371,27 € vs el nuevo 13.454,51 se observa que ha habido un incremento de 1.083,24 €.

Recursos materiales

Para el desarrollo del proyecto se ha utilizado un portátil MacBook Pro (13", 2016) con capacidad de almacenamiento de 500 GB. Por otro lado, los recursos software usados son gratuitos, por lo tanto, no implicarán un coste extra en el desarrollo.
Sin embargo, este dispositivo tiene una vida útil limitada, consecuentemente, es necesario calcular el coste de la amortización.

Amortización = (Precio total * 5 meses) / (años vida útil * 12 meses)

Tabla 6: Coste recursos humanos. Fuente: Elaboración propia

Recurso	Precio (€)	Vida útil (años)	Amortización (€)
MacBook Pro (2016)	1.700	8	88,54

Recursos indirectos

Es necesario contemplar los recursos que se emplean durante todo el proceso, como por ejemplo acceso a Internet o el coste eléctrico.

Tabla 7: Coste recursos indirectos. Fuente: Elaboración propia

Recurso	Precio	Unidades	Coste (€)
Consumo energético	0,29 € / kWh	**510,25 h**	42.17
Internet	60 € / mes	**8 meses**	480
Espacio desarrollo	350 € / mes	**8 meses**	2.800
TOTAL	-	-	**3.322,17**

Imprevistos

En la sección anterior se detallaron los posibles imprevistos que podrían surgir al desarrollar este proyecto. Sin embargo, en caso de que estos ocurrieran no supondrán un coste económico extra, es por eso por lo que no se consideran en este apartado.

Aun así, pueden suceder otros imprevistos relacionados con los recursos materiales previamente mencionados. Para estimar el coste se supondrá el peor caso, es decir, el reemplazo del recurso. Aun siendo la posibilidad baja, es necesario contemplar el riesgo.

Tabla 8: Costes imprevistos. Fuente: Elaboración propia

Recurso	Precio (€)	Probabilidad (%)	Coste (€)
MacBook Pro (2016)	1.700	10	170

Contingencias

El coste de contingencia es la cantidad añadida al presupuesto base, para poder cubrir y solucionar imprevistos. Para este proyecto, el valor se fijará en un 10%.

Tabla 9: Costes contingencias. Fuente: Elaboración propia

Tipo de coste	Coste (€)
Coste de personal	13.454,51
Costes materiales	1.700
Coste indirecto	3.322,17
Total	**18.476,68**
Contingencia total	**1.847,67**

Presupuesto Final

En la siguiente tabla se resumen todos los costes previamente establecidos. En esta, se puede observar el que será el presupuesto final para el proyecto.

Tabla 10: Presupuesto Final. Fuente: Elaboración propia

Tipo de coste	Coste (€)
Coste de personal	**13.454,51**
Coste material	88,54
Coste indirecto	**3.322,17**
Contingencia	1.847,67
TOTAL	**18.712,89**

Finalmente, con relación al prepuesto total del proyecto, si comparamos el coste inicial 18.186,85 € vs. el nuevo 18.712,89 € se observa que ha habido un incremento de 526,04 €.

8.2. Control de gestión

Con el objetivo de mantener una correcta gestión durante el desarrollo, cada vez que se termine una etapa, o *sprint*, se actualizará el presupuesto según las horas que hayan sido trabajadas y los imprevistos. Esta actualización se llevará a cabo mediante las siguientes fórmulas.

- **Desviación de horas estimadas por tarea**
  (horas estimadas - horas reales) x Coste estimado

- **Desviación del coste según las horas trabajadas por tarea**
  (horas estimadas - horas reales) x Coste real

- **Desviación del coste en recursos humanos por tarea**
  (coste estimado - coste real) x Horas reales

- **Desviación causada por imprevistos**
  costes imprevistos estimados - coste imprevistos reales

- **Desviación total de horas**
  Horas totales estimadas - horas totales reales

9. Sostenibilidad

Durante las asignaturas cursadas en el grado se ha animado al alumnado a reflexionar sobre las diferentes dimensiones que encontramos en la sostenibilidad, especialmente dirigidas al sector tecnológico.

Personalmente, el aspecto sobre el que tengo más conocimiento es el ambiental, ya que resulta más común encontrarse con información sobre este y suele estar presente en nuestro día a día. Además, en varias ocasiones hemos participado en talleres de reciclaje, así como otras actividades que han permitido que crezca el interés e incremente la concienciación sobre este aspecto.
Sin embargo, tanto el ámbito social como el económico resultan cruciales para poder determinar la dimensión de un proyecto. Por lo tanto, debemos asegurarnos de que trabajamos de forma ética, con software sostenible y accesible para todos. Finalmente, la dimensión económica se basa en estudiar y analizar el mercado y proporcionar un correcto análisis y presupuesto de costes.

9.1. Control de gestión

¿Has estimado el impacto ambiental que tendrá la realización del proyecto? ¿Te has planteado minimizar el impacto, por ejemplo, reutilizando recursos?
Se contempla el impacto ambiental que pueda tener la selección de determinados componentes hardware. Estos se eligen en función de su vida útil, así como el impacto ambiental que tienen.
En relación con los imprevistos planteados, más concretamente, cuando un componente no funcione correctamente, se optará por la reparación por encima del reemplazo directo, con el objetivo de minimizar el impacto ambiental.

¿En qué mejorará ambientalmente tu solución a las existentes?
Se pretende dar solución a una aplicación software. El objetivo es proporcionar una estructura hardware que sea eficiente y responsable, utilizando componentes con consumo mínimo.
Además, la aplicación software que esta solución respalda, permitirá que usuarios pueden fácilmente hacer aportaciones y contribuir a ONG responsables con el medio ambiente, por lo tanto, indirectamente podría mejorar la dimensión ambiental.

9.2. Dimensión económica

¿Has estimado el coste de la realización del proyecto (recursos humanos y materiales)?
Si, se ha hecho un estudio tanto sobre el coste de los recursos humanos como los materiales. En este, se refleja el impacto que tendrá el proyecto.

¿En qué mejorará económicamente tu solución a las existentes?
Podría extrapolarse la plataforma hardware elegida a otros servicios similares, o incluso juntar varias entidades con el mismo objetivo. De esta forma compartirán recursos y, por lo tanto, se reduciría el coste.

9.3. Dimensión económica

¿Qué crees que te va a aportar a nivel personal la realización de este proyecto?
Gracias al desarrollo de este proyecto aprenderé a hacer un estudio completo para determinar una plataforma hardware. Personalmente, el proyecto me aportará un gran conocimiento técnico.
En el ámbito personal, me enorgullece poder formar parte de un proyecto de estas características, ya que ayudará a la creación de una plataforma que fomentará y facilitará que usuarios o empresas puedan hacer aportaciones a una inmensa cantidad de asociaciones benéficas.

¿En qué mejorará socialmente (calidad de vida) tu solución a las existentes?
Este estudio hardware, como se ha comentado anteriormente, permitirá que un equipo de desarrolladores pueda implementar una aplicación.

¿Existe una necesidad real del proyecto?
Si, ya que es crucial para una plataforma software, tener una estructura hardware donde poder desarrollarse.

10. Entorno hardware

Se marca como objetivo de la memoria proporcionar un entorno de desarrollo, que será utilizado en una aplicación que llevará a cabo una gran cantidad de accesos a la base de datos. Para ello, será necesario determinar cuál es el mejor procesador para dicha tarea.
Por eso a continuación se estudian las diferentes arquitecturas más populares en este momento. Se comprará la arquitectura x86/x64 de una CPU y GPU.

Un procesador es el componente más importante de un PC, así como el cerebro lo sería en el cuerpo humano, ya que dicta y coordina todas las tareas que se realizan en dicha máquina. Por eso, también es comúnmente conocido como CPU o, en otras palabras, la Unidad Central de Procesamiento.

La forma en la que un ordenador trabaja es mediante instrucciones, en otras palabras, el ordenador ejecutará órdenes. Sin embargo, existen varios métodos o enfoques para llevar a cabo este proceso, y el diseño elegido afectará directamente a la velocidad a la que el procesador es capaz de funcionar. La rapidez a la que el ordenador trabaja equivale a la frecuencia de funcionamiento, la cual se mide en giga hercios (GHz). La frecuencia del procesador permite sincronizar todo el sistema, y dictará a su misma vez la velocidad a la que trabajará todo el ordenador.
Para una mejor comprensión, supongamos que el ordenador ejecuta una instrucción por ciclo. Si se duplica la frecuencia, técnicamente se doblará la cantidad de trabajo que se ejecuta en el mismo tiempo. De ese modo, se diseñaron los primeros procesadores, pero a medida que la tecnología ha avanzado, se ha conseguido que varias instrucciones puedan completarse en un solo ciclo, por lo que la frecuencia ya no es el parámetro más importante para determinar la velocidad de un procesador. (1)

Como todo componente, un procesador tiene limitaciones físicas que deberán ser consideradas cuando quieran aplicarse mejoras o innovaciones en este. El impedimento más grande suele ser el consumo energético. Con los descubrimientos que se han hecho desde entonces, no resulta demasiado complejo mejorar la velocidad y conseguir que un procesador con menor frecuencia supere a uno que con una sincronización superior.

Los cambios que se introducen en la arquitectura como, por ejemplo, añadir bloques de cálculo, desarrollo de nuevas instrucciones… van frecuentemente ligados de mejoras en el software, para que en su conjunto optimicen el rendimiento total.

Una de las primeras mejoras fue añadir más núcleos, esto se conseguía mejorando la técnica de fabricación y creando transistores cada vez más pequeños.
Este cambio solo implicaría conectarlos entre sí, y sin embargo, se conseguía impulsar el rendimiento del procesador exponencialmente. El espacio libre ocasionado por la reducción

del tamaño de los transistores daba lugar a la posibilidad de integrar componentes externos junto al procesador…
Esta evolución finalmente ha dado lugar a los SOC (*System On a Chip*) que son capaces de integrar controladores de memoria, tarjeta gráfica…

La forma en la que se diseñan los procesadores, la interconexión entre elementos y, por último, el diseño final son lo que se conoce como el estudio de la arquitectura del procesador.

10.1. Set de instrucciones RISC vs CISC

La arquitectura RISC (*Reduced Instruction Set Computer*) se caracteriza por trabajar con un conjunto reducido de instrucciones.
Su funcionamiento se basa en ejecutar una sola instrucción por ciclo, en el cual se cargarán todos los valores necesarios, que posteriormente modificará o consultará dependiendo de la necesidad. Cuando se termine este proceso, se volverá a repetir con la siguiente instrucción a ejecutar. (2)
Esta arquitectura necesita trabajar con una RAM mayor, ya que para cada ciclo de reloj deben cargarse nuevos datos. Además, al ejecutar un programa paso a paso, no se sobrecalienta el ordenador, puesto que el procesador no se excede.

Por otro lado, CISC (*Complex Instruction Set Computer*) trabaja con un conjunto más complejo de instrucciones. Se diferencia de la otra arquitectura en que es esta, es capaz de ejecutar instrucciones mucho más complejas, que pueden necesitar más de un ciclo para completarse.

Por lo tanto, optimizar el código y agrupar varias instrucciones para generar una aún más compleja, esta arquitectura siempre ejecutará menos pasos para completar una tarea, aunque esto le suponga un mayor coste energético. Mientras que la arquitectura RISC, ejecutará un mayor número de pasos, pero estos serán más simples y por lo tanto requerirán menos energía, por lo que el ordenador generará menos calor.

En cualquier caso, no existe una opción correcta, ya que, si este fuese el caso, todas las compañías utilizarían la misma arquitectura, aunque también es cierto, que RISC apareció a principios de 1980 como un rediseño a la arquitectura previa, CISC.

10.2. Unidad Central de Procesamiento

Actualmente, encontramos dos arquitecturas muy utilizadas que se han convertido en la elección más común para las computadoras. Estas son la arquitectura de 32 bits, también conocida como x86, o la de 64 bits, conocida como x64. Además, trabajarán con el conjunto de instrucciones CISC.

La arquitectura determinará la forma en la que se almacena la información, el tamaño de las palabras. Esto conllevará que los datos se guarden en conjuntos de 32 o 64 bits. Generalmente, cuanto mayor es el tamaño de la palabra, más rápida será la obtención de información, es por eso, que la mayoría de los ordenadores que actualmente se encuentran el mercado emplean un conjunto superior de bits. Sin embargo, un tamaño superior también tiene sus propias desventajas.

La necesidad de evolucionar a un conjunto de datos mayor está altamente relacionada con la capacidad máxima de gestión de una memoria RAM. Una arquitectura basada en 32 bits podrá únicamente sacar provecho de aproximadamente de 4 GB, y por otro lado, una arquitectura de 64 podría trabajar hasta con 16 Exabytes, lo cual ha sido un salto enorme y sugiere que estos sistemas nos acompañaran durante un largo tiempo. (3)
Por supuesto, las posibles exigencias o prestaciones necesarias varían en función del uso que se le dé al equipo. Además, estos sistemas son retro-compatibles, así que una arquitectura mayor siempre podrá gestionar programas diseñados para un sistema inferior. Este mismo proceso no funcionará al revés, debido a la falta de integración y capacidad de gestión de la memoria.

A continuación, se cubrirán los elementos principales que forman una CPU. (4)

Banco de registros

Un registro es la unidad más sencilla de un procesador y la encargada de almacenar información. Se caracterizan por ser rápidos a pesar de solo poder almacenar una porción pequeña de datos.
Existen diferentes tipos de registros, estos tienen asignadas tareas especiales.
El primero es el registro **AC** (*Accumulator*), que guarda los resultados de los cálculos.
Los registros **IR** (*Instruction Register*) y **MAR** (*Memory Address Register*) se encargan de guardar la dirección, es decir, la posición en memoria de la instrucción y datos a ser procesados, respectivamente.
El registro **MDR** (*Memory Data Register*) almacena la información que está siendo procesada.
Y finalmente, **PC** (*Program Counter*) guarda la dirección de la instrucción que se ejecutará a continuación.

Unidad Lógico-Aritmética

La ALU (*Arithmetic Logic Unit*) es la unidad responsable de hacer todos los cálculos. Utilizará puertas lógicas para hacer todo tipo de operaciones como comparación, suma, sustracción ….

Esta recibe dos operandos que serán sometidos a una operación. Así como el código de operación para saber que calculo aplicar sobre estos. Finalmente, la señal de estado determinará si se puede recibir nueva información, o, por otro lado, la ALU está trabajando en un cálculo del ciclo anterior.

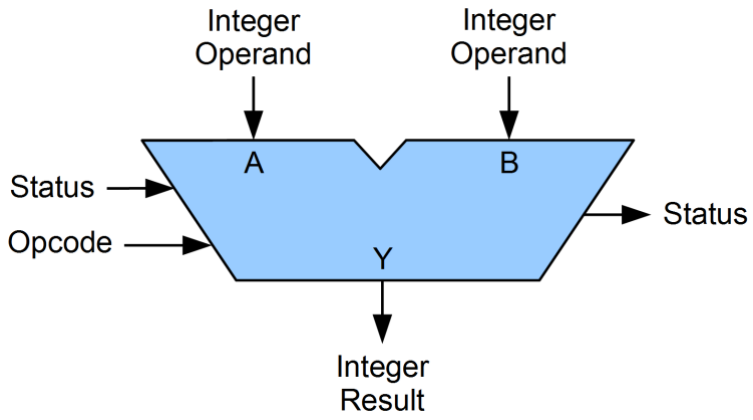

Figura 2: Representación ALU. Fuente: (15)

Unidad de Control

A menudo se habla de la CPU como el cerebro de un ordenador, sin embargo, la Unidad de Control es el componente, dentro del procesador, que decodifica las instrucciones y controla como han de funcionar el resto de las unidades en consecuencia. Además, también se encuentra un pequeño reloj que determina la velocidad a la que se ejecutan los cálculos del procesador, en otras palabras, la frecuencia de la CPU.

En la siguiente imagen, se representa de forma básica un diagrama del funcionamiento de un ordenador.

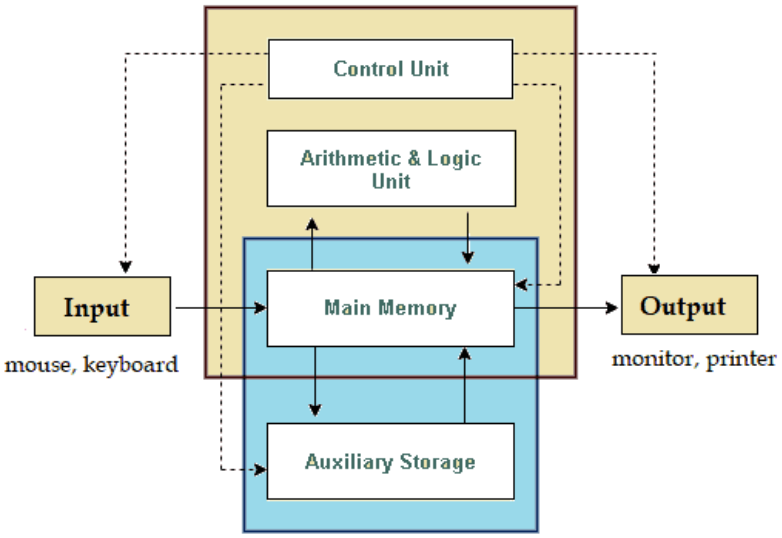

Figura 3: Diagrama de un Computador. Fuente: (16)

La unidad obtendrá información de la memoria que es donde se encontrará la información del programa a ejecutar, el conjunto de instrucciones. Seguidamente, informará a la ALU, la cual se encargará de hacer todos los cálculos necesarios. Cabe recalcar que la Memoria y ALU están conectadas, ya que esta última debe poder escribir y leer datos.

Una vez se haya completado la instrucción se producirá la salida, en caso de ser necesario, y se comunicará a los puertos exteriores (Output), así como se haría con los datos que se recogen (Input).

10.3. Unidad de Procesamiento Gráfico

La GPU es la unidad de procesamiento gráfico, el "cerebro" de una tarjeta gráfica. Dicho componente es frecuentemente designado a completar cálculos complejos en el ámbito de los videojuegos o programas con alto procesado en imágenes y/o videos. (5)

Cuando un ordenador cuenta con ambos una CPU y GPU, el rendimiento de este puede verse exponencialmente incrementado. Esto se debe a que las tareas más intensas o que requieren cálculos más complejos son asignados a la GPU, que como se ha observado, trabaja de forma excelente bajo este contexto.
Y, por otro lado, las tareas más sencillas las llevará a cabo la CPU. La pregunta entonces reside en cuál es la razón detrás de esta diferencia.

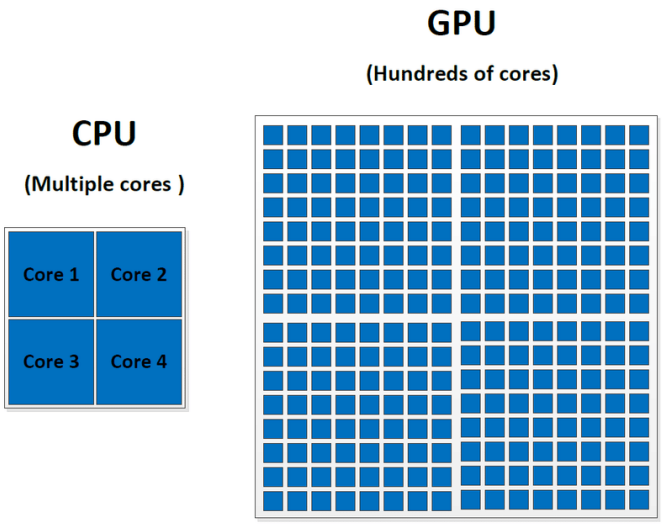

Figura 4: Estructura de una CPU vs GPU. Fuente: (6)

Una CPU está formada por un número reducido de núcleos, y diseñada para completar tareas secuenciales, mientras la GPU cuenta con miles de núcleos que trabajan en paralelo.

Aun así, la GPU no remplaza por completo a la CPU, ya que esta última es el centro de un ordenador y, por lo tanto, delega las tareas, entre ellas, las que ejecutará la GPU.
Tanto un componente como el otro presentan ciertas limitaciones. Es cierto que la GPU supera a la CPU en cuanto número de núcleos, pero, sin embargo, estos son menos eficaces y es por eso por lo que ejecutarán tareas más pequeñas. A su misma vez, este componente es realmente eficiente al trabajar en paralelo, lo que le permitirá completar varias tareas a la vez o repartir la carga de una entre varios núcleos.

Una vez más existen una larga cantidad de modelos y variaciones en la estructura de una GPU, pero a continuación se pretende exponer aquellos componentes "básicos" que suelen encontrarse en todas.

Para facilitar la comprensión de los diferentes componentes, se ha tomado como modelo el esquema de la arquitectura de una GPU de la familia Fermi de Intel. (7)

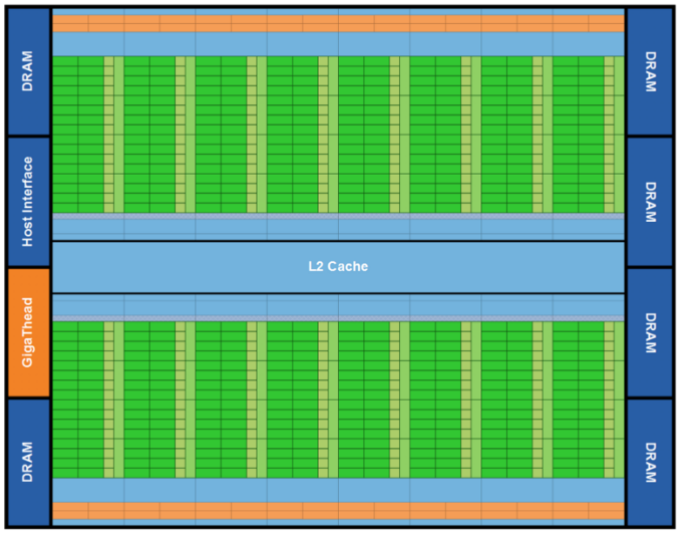

Figura 5: Arquitectura de una GPU Fermi. Fuente: (7)

Unidades Streaming Multiprocessor

En este modelo se encuentran 16 unidades SM. Estas están representadas como un rectángulo vertical verde y forman el "corazón" de la GPU.

Cada unidad cuenta con 32 Cores. Se puede observar cómo estos son de un tamaño considerablemente inferior al Core tradicional que se encuentra en una CPU, pero en conjunto consiguen proporcionar un alto rendimiento. Además, también cuentan con un banco propio de registros, así como una memoria compartida y una caché.

Una unidad SM ejecuta instrucciones de un grupo consecutivo de *threads*[1], llamado *warps*.
El módulo *Dispatch U*nit se encarga de asegurar que los threads se ejecuten, y el *Warp Scheduler* gestiona los diferentes *warps* y cambia el contexto entre estos para un correcto desarrollo.

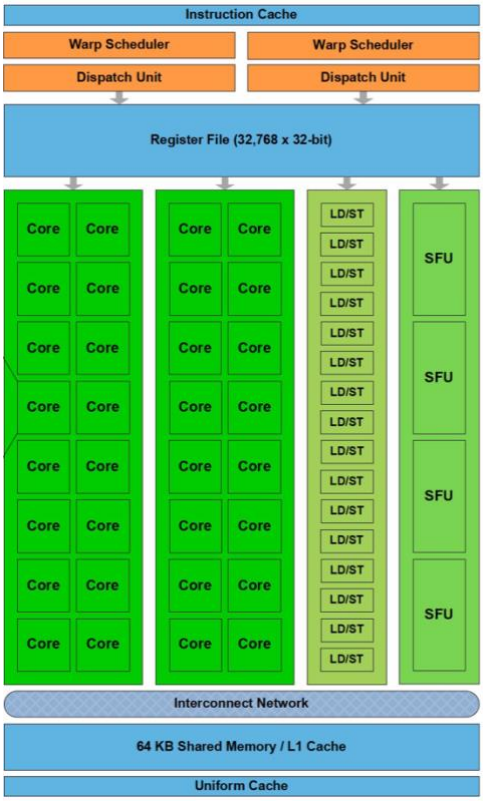

Figura 6: Fermi Streaming Multiprocessor (SM). Fuente: (7)

Memoria gráfica

La memoria gráfica no suele estar integrada con la GPU, pero, sin embargo, tiene un papel muy importante en el rendimiento de esta.
La GPU trabaja con datos grandes que requieren un procesado muy elevado, y para poder gestionar estos cálculos se necesita poder almacenar la información temporal en algún lugar. Aquí es donde se introduce la memoria gráfica, ya que la capacidad de almacenamiento de los registros es muy limitada. Este componente es "opcional", puesto que la GPU puede funcionar correctamente sin este, pero en esencia, esta colaboración permite que se guarden datos mientras se computan otros cálculos. De esta forma, cuando se quiera recurrir de nuevo a la información previamente procesada, esta estará almacenada en la Memoria Gráfica, y la GPU podrá recurrir a esta sin tener que volver a calcularla.

[1] **Thread:** Secuencia de instrucciones programadas que puede ser gestionada por un sistema operativo.

Bus de memoria

Este componente tiene una alta implicación con la velocidad utilizada por la memoria gráfica. El bus determinará el ancho de banda disponible para la comunicación entre memoria y GPU, donde un bus mayor permite una transmisión de datos más rápida, pero, sin embargo, a un precio mayor.

Finalmente, como se ha comentado anteriormente, la GPU juega un rol importante en los programas de edición, juegos de alto rendimiento… Por esa razón, a continuación, se mencionarán una serie de componentes diseñados para facilitar este trabajo.

Shaders

Estos motores de sombreado se encargan de la transformación de color, sombras y otros efectos relacionados con la iluminación.

Unidades de rasterizado y texturizado

Estas unidades, también conocidas como ROP o TMU respectivamente, ayudan en el procesado de imágenes.
La primera transforma una imagen expresada en formato vectorial a un conjunto de píxeles, los cuales son directamente transmitidos a la pantalla.
La segunda, aplican las texturas a la geometría, elevando la calidad y realidad de las imágenes.

11. Computación GPU

Las GPU se han utilizado comúnmente como un componente extra que acompaña a las CPU, es decir, al procesador principal de una máquina. Este componente añadidoacelera y permite obtener resultados más rápidos para operaciones que hacen un uso intensivo de la memoria, así como para generar y renderizar imágenes y/o videos.

Con los avances tecnológicos que se han visto durante estos últimos años, actualmente es posible usar computadoras, cuyo objetivo es realizar operaciones genéricas, basadas en GPU. (8)

11.1. Ley de Moore

Gordoon Moore en 1965 observó una tendencia en los microcontroladores, que describió de la siguiente forma:

"El número de transistores por unidad de superficie en circuitos integrados se duplicará cada año" (9)

Posteriormente, esta frase, que terminó definiendo como se entiende hoy en día la computación, fue actualizada y Moore rectifico sus palabras explicando que esta duplicación ocurriría cada dos años.

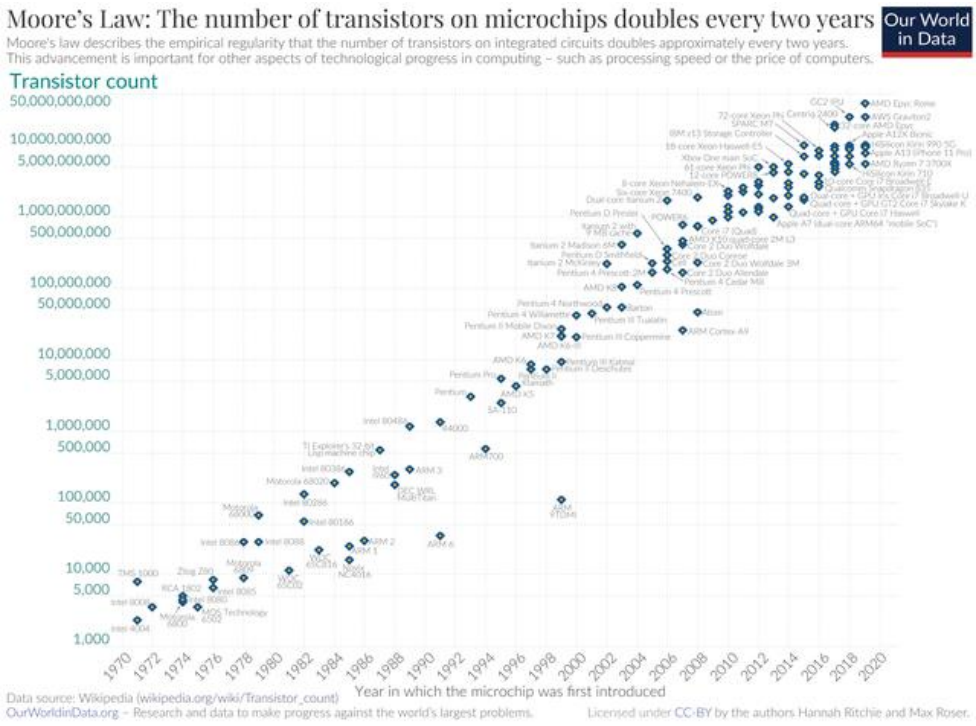

Figura 7: Ley de Moore. Fuente: (20)

Esta ley tiene una alta implicación en el hardware, debido a que para poder duplicar el número de transistores[2] se debe reducir el tamaño de estos.
La litografía es el término utilizado para hacer referencia a la relación entre cantidad y tamaño. Por lo tanto, una menor litografía implicará que dicho procesador tenga transistores más pequeños y en consecuencia se incluya un mayor número de estos en el mismo espacio, donde previamente solo entrarían la mitad.

Esta continua evolución tiene una gran implicación para la eficiencia energética y la velocidad. Por un lado, cuando se reduce el tamaño (medido en nanómetros) de un transistor, la cantidad de electrones que circulan por cada uno de estos también pasa a ser inferior. Consecuentemente, el transistor será capaz de cambiar de estado consumiendo menos energía. Así pues, no solamente mejora la eficiencia energética, sino que un procesador de 14 nm será capaz de hacer el mismo trabajo que uno de 7 nm, pero gastando menos energía, o, en otras palabras, se mejora la potencia de procesamiento por cada vatio consumido.

Por consiguiente, el procesador será capaz de trabajar con una mayor cantidad de estados al mismo tiempo, que llevará a una potencia superior y, por lo tanto, una mayor velocidad. Esto significa, que se incremente la cantidad de instrucciones que se ejecutan por ciclo. (10)

11.2. Funcionamiento GPU

En el apartado anterior donde se habla de la Unidad de Procesamiento Gráfico se describen los componentes principales de una GPU, así como la comunicación entre estos.

De forma sencilla, una CPU se orienta a ejecutar tareas lineares a una gran velocidad, mientras que la GPU se orienta a la paralelización, que consecuentemente permite ejecutar un mayor número de tareas al mismo tiempo o una sola, pero en un tiempo reducido.

Una CPU tiene un número menor de *Cores,* ya que estos no necesitan colaborar entre sí, ya que trabajarán en tareas independientes.

[2] **Transistor**: El elemento más pequeño que puede encontrarse en un circuito electrónico.

A continuación, se expone el ejemplo de una simple suma de los componentes contenidos en un *array*. Se comparará como cada una de arquitecturas gestiona esta operación.

Una CPU cogerá el primer elemento de la secuencia y lo sumará al siguiente, esto serían los números 13 y 27 en el ejemplo que puede observarse en la imagen.
Una vez se obtiene el resultado, este se sumará al siguiente valor y así sucesivamente hasta obtener el resultado final.

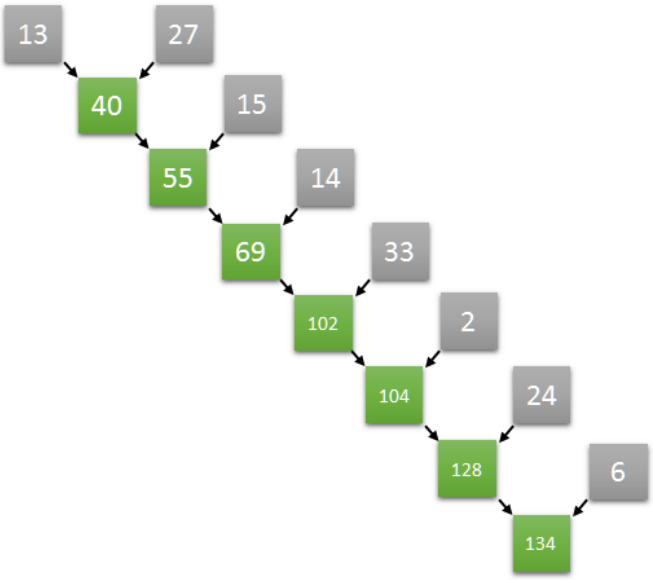

Figura 8: Representación de una suma linear. Fuente: (17)

Por otro lado, una GPU transformará este cálculo en una secuencia de operaciones paralelas. En el caso de una suma, el orden de los factores no altera el producto y esta propiedad permitirá que se hagan sumas de dos valores al mismo tiempo, los resultados de cada una de estas se sumarán posteriormente, y así continuamente hasta obtener el resultado esperado.

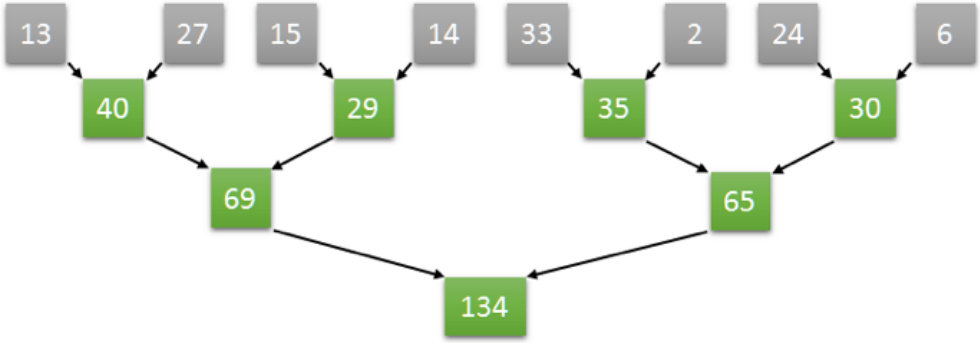

Figura 9: Representación de una suma en paralelo. Fuente: (17)

Si se comparan ambos ejemplos, la misma operación se realizó en 8 etapas diferentes en el caso de la CPU. Y, sin embargo, en 3 para la GPU.

Al llevar este ejemplo a un caso real, los *Cores* de una GPU distribuirían las operaciones de la siguiente forma. Para una mejor compresión se denominará cX a los Cores 0 a 3, por ejemplo, c0, c1, c2…

En el primer ciclo de reloj a cada *Core* se le asigna una operación. La suma de los valores 13 y 27 queda a cargo de c0, los valores 15 y 14 los gestionará c1… y así sucesivamente.
En el siguiente ciclo ya se dispondrá de los resultados, pero cabe recalcar que ahora solo será necesario sumar 4 valores, que quedarán a cargo de c0 y c1. Una vez se ha terminado, c0 hará el último cálculo en el tercer ciclo y será cuando se muestre el valor final.

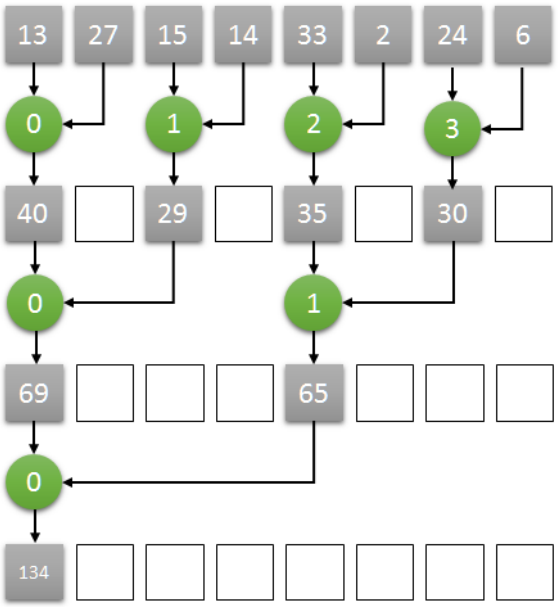

Figura 10: Reducción de una suma en paralelo en una GPU. Fuente: (17)

11.3. Memoria GPU

La colaboración entre GPU y memoria tiene una gran implicación en el rendimiento de esta. Debido a que se pueden encontrar hasta cientos de *Cores* en este tipo de procesadores, estos se dividen por grupos, ya que en caso contrario implementar la colaboración entre todos sería demasiado costoso.

Finalmente, siguiendo la arquitectura de la familia Turing, los *Cores* se organizarán en diferentes SM (*Streaming Multiprocessors)* donde cada uno tiene una pequeña memoria caché compartida (L1 Caché). Además, todos los SM comparten una memoria mayor (L2 Caché) que funciona a una velocidad más reducida.

En caso de necesitar datos que no se encuentran en la L2, se accederá a la memoria DRAM principal de la GPU.

Figura 11: Arquitectura Turing. Fuente: (17)

Pero ¿Cuáles son los puntos fuertes y débiles de la GPU?, y ¿En qué se diferencia de una CPU?

Como se ha podido observar, una GPU destaca por su gran capacidad de paralelizar tareas. Sin embargo, no todas las operaciones que un ordenador ejecuta pueden dividirse en subtareas, es decir, cada programa tiene un nivel de paralelización posible diferente. Pero este procesador será una buena elección para todas aquellas aplicaciones que puedan aprovecharse de esta característica.

Para poder exprimir al máximo el rendimiento que puede ofrecer una GPU idealmente todos los datos relevantes deben poder almacenarse en la memoria DRAM de esta. En caso contrario, no podrá procesarse la información de forma simultánea.

Las posibilidades que ofrece un procesador gráfico son múltiples y por supuesto, cada vez hay más campos donde puede aplicarse. Entre estos, destacan los escenarios como Inteligencia Artificial, diseño gráfico o incluso estudios de previsiones meteorológicas y sísmicas.

Por otro lado, como respuesta a la gran cuestión sobre si una GPU podrá superar o reemplazar a una CPU, el CEO de la compañía NVIDIA en 2017, hizo la siguiente declaración:

"As advanced parallel-instruction architectures for CPU can be barely worked out by designers, GPUs will soon replace CPUs" (11).

La idea que Jensen Huang quería transmitir con esta frase es que la velocidad en la que la tecnología avanza dificulta que los diseñadores sean capaces de mantener el ritmo y, por lo tanto, arquitecturas o instrucciones más complejas que trabajan en paralelo, no puedan ser adaptadas en una CPU. Esta falta de integración llevaría a que las GPU reemplazaran a las CPU, ya que estas primeras si serían capaces de llevar a cabo grandes instrucciones en paralelo, debido al gran número de núcleos que las componen.

Sin embargo, se ha llegado a la conclusión que una GPU no reemplazará al procesador en un ordenador, pues para que pudiera darse esa situación habría que tener en cuenta diferentes factores. En caso de que se sustituyeran por completo, el gasto energético se elevaría de forma exponencial y también supondría un problema en el intercambio de datos, ya que habría que encontrar un método que permita proporcionar la información a la misma velocidad que la GPU la consumiría. Es por eso por lo que se utilizan las GPU como aceleradores o *boosters*, es decir, componentes utilizados para incrementar el rendimiento de una aplicación o reducir el tiempo de cálculo. Es por eso, que hoy en día, la gran mayoría de procesadores cuentan con una pequeña GPU integrada.

12. Bases de datos

Una base de datos es una herramienta que se utiliza para gestionar y almacenar de forma organizada diferentes tipos de información.
Con el avance de la tecnología, las necesidades de los usuarios evolucionan, por lo que hoy en día encontramos varías propuestas para proporcionar este servicio de recopilación, organización y relación de datos. Entre estas, existen dos métodos que han terminado convirtiéndose en el estándar actual. Las bases de datos relaciones (SQL) o las no relaciones (No SQL).

La diferencia principal entre ambas es que la primera utiliza una estructura relacional, y además, cuenta con un esquema predefinido que se utiliza para almacenar y gestionar la información. Por otro lado, No SQL usa esquemas dinámicos, que no requieren una estructura fijada para los datos almacenados, es decir, ofrece mayor flexibilidad.
SQL ha sido durante mucho tiempo la elección por defecto, debido a su versatilidad y estructura. Sin embargo, esta última característica es a la vez su mayor desventaja, ya que puede ser demasiado restrictiva.
Todos los datos almacenados deben seguir la misma estructura, y requiere de un trabajo previo bastante elevado y exponencial, en relación con la cantidad de datos con los que se trabaje. Además, como estas relaciones se fijan al inicio, un cambio posterior a la creación implicara que el proyecto deba ser replanteado y probablemente sea necesario volver a empezar.

12.1. Bases de datos relacionales

Una base de datos SQL recopila la información y la organiza en campos, que se relacionan entre sí. Para poder entender de forma más fácil los diferentes componentes que dan lugar a una tabla, se utilizará el ejemplo que hay a continuación. (12)

Tabla 11: Ejemplo tabla SQL. Fuente: Elaboración propia

ISBN	Título	Editorial	Autor	Precio
978-3-16-148410-1	TituloLibro1	Editorial1	Autor1	15 €
978-3-16-148410-2	TituloLibro2	Editorial2	Autor2	20 €
978-3-16-148410-3	TituloLibro3	Editorial3	Autor3	32 €

Se entiende como **tabla** o relación, el conjunto de tuplas que comparten los mismos atributos, es decir, un conjunto de filas y columnas. Para el ejemplo, se podría nombrar la tabla como "Libros", ya que esta nos mostrará diferentes atributos o características de un libro.

Cada **columna** representará un atributo o categoría, típica del objeto que se está tratando en la tabla, en este caso, diferentes atributos de los libros *(ej. Título, Autor, Precio…).*
Sin embargo, cada una de las columnas puede representar un tipo de información diferente, y por eso resultará importante determinar qué tipo de valor es cada atributo.
Finalmente, las **filas** o tuplas/registro, es el conjunto de datos que representan un objeto, es decir, una instancia de la tabla. En el anterior ejemplo cualquiera de las filas que aparece en la figura 9, es considerada un registro.

Una vez alcanzado este punto, resulta fácil completar y añadir información a las tablas. No obstante, cuanta más información, más probabilidad hay de que se repitan algunos datos.
Volviendo al ejemplo anterior (figura 9), no hay ningún inconveniente con que por ejemplo el Libro1 y el LibroX compartan el mismo precio. Sin embargo, si ambos libros son escritos por el mismo autor/a, resultará más complicado determinar de qué libro se está hablando, ya que ese atributo no es único, y será necesario, o bien proporcionar el Autor+Título o, una serie de atributos que en su conjunto sean únicos. Esto se conoce como clave primaria o *Primary Key*.

Cuando se busca simplificar este proceso, a menudo se añade una columna a la tabla, un nuevo atributo al objeto que será único, que se podrá utilizar como identificador. En el caso de los libros, este atributo se conoce como ISBN, un código único para cada libro. En el caso de un ciudadano se emplea el DNI y si no hay un campo ya establecido, simplemente se puede generar una nueva columna que se utilizará como identificador y será única para cada tupla.

Tabla 12: Ejemplo tabla SQL 2. Fuente: Elaboración propia

ISBN / ID	Título	Editorial	Autor	Precio
978-3-16-148410-1	TituloLibro1	Editorial1	Autor1	15 €
978-3-16-148410-2	TituloLibro2	Editorial2	Autor2	20 €
978-3-16-148410-3	TituloLibro3	Editorial3	Autor3	32 €

Relación de tablas

Como se ha mencionado previamente, es necesario hacer un diseño previo de la estructura de las tablas, es decir, como se van a relacionar estas entre sí.

Para poder comprender mejor este concepto, se expandirá el ejemplo anterior. Para diseñar la organización de una universidad, sería necesario determinar aquellas tablas importantes, por ejemplo, Alumnado, Profesorado, Asignatura, Libros.

Una vez se termina la primera fase, donde se han determinado las clases que formarán parte de la organización, se pasará a la segunda etapa, donde se relacionan las tablas hasta de tres maneras diferentes.
La primera es la relación uno a uno, donde por cada registro de una tabla, hay un registro en otra tabla diferente. La segunda, es una a varios, donde por varias tuplas de una clase, habrá una instancia en otra. Finalmente, queda la relación varios a varios, que es aquella establecida entre varias entidades de cada una de las tablas.

Un ejemplo de cómo podrían organizarse en el ejemplo que se ha presentado, sería el siguiente:

- Una Asignatura es instruida por un solo Profesor.
- Un Profesor puede instruir varias Asignaturas.
- Una Asignatura es impartida a varios Alumnos
- Un Alumno puede estudiar más de una Asignatura
- Una Asignatura está basada en un Libro
- ….

Las reglas que se definan inicialmente determinaran como se relacionan las tablas.
Es más, de una de estas relaciones puede crecer una nueva tabla, como por ejemplo ocurre cuando se relacionan la tabla Asignatura-Alumnos, y aparece la tabla Grupo, que podría representar un conjunto determinado de Alumnos que estudian una determinada Asignatura.

Para poder llevar a cabo este tipo de relaciones, se utilizará el identificador de cada una de las tablas, y la nueva relación que se ha creado tendrá como *Primary Key* ambos valores. Por lo tanto, *IDgrupo = (IDalumno, IDasignatura)*.

SQL statements

Una base de datos relacional organiza y almacena información. Sin embargo, para poder añadir, eliminar o hacer algún cambio existe un conjunto instrucciones que permitirán que el usuario interactúe con esta. (13)
Para el estudio de este proyecto, se medirá el funcionamiento e impacto de las siguientes instrucciones.

Insert

Esta instrucción permite añadir una o más filas a una tabla.
La sintaxis requiere que primeramente se especifique aquella tabla a la cual queremos añadir información, y a continuación todas aquellas columnas que se desean completar. A continuación, por cada columna se añadirá, manteniendo el mismo orden, el valor para cada una de estas.

```
INSERT INTO nombreTabla (columna1, columna2 …)
    VALUES (valor1, valor2 …)
```

En caso de que quiera añadirse más de una fila para la misma tabla deberá añadirse una coma al final del paréntesis que agrupa los valores de la primera fila, y abrir un nuevo paréntesis donde se añadirán los nuevos valores.

Update

Esta instrucción permite modificar los valores de una columna para una o varias filas.
Primeramente, se especificará una vez más, el nombre de la tabla cuya información quiera ser modificada y a continuación el nuevo valor que se asignará a la columna correspondiente. Se pueden añadir tantas modificaciones como sea necesario, simplemente es necesario que estas se separen mediante comas.

```
UPDATE nombreTabla
SET columna1 = valor1, columna2 = valor2, …
WHERE filtro = valor
```

Delete

Esta instrucción permite eliminar una o varias filas de una tabla.
Para poder eliminar información el primer paso será establecer la tabla que se desea modificar, y a continuación especificar la condición que determinara las filas que serán eliminadas.

```
DELETE FROM nombreTabla
WHERE filtro = valor
```

12.2. Bases de datos no relacionales

Las bases de datos no relacionales nacen con la introducción del *BigData*[3].
La aparición de las redes sociales, entre otras, implica que se genere una cantidad desmesurada de contenido por minuto. El usuario de dichas plataformas debe poder acceder toda la información de forma rápida y desde cualquier lugar. Esta nueva dinámica requiere una nueva gestión más rápida y de un tamaño mucho más elevado de almacenamiento de datos.
Como se estudia en el apartado anterior, en bases de datos relacionales, las columnas nos permitirán organizar los atributos de un objeto de forma más eficiente, y por cada conjunto de columnas único aparecerá una fila o instancia.
Sin embargo, en bases de datos no relacionales estos no se agruparán de la misma manera ni utilizarán lenguaje SQL, de ahí proviene dicho nombre. Para que un usuario pueda comunicarse con una base de datos NOSQL usará lenguajes propios de los gestores de BBDD como Cassandra, MongoDB…. (12)

Almacenamiento de información

Acudiendo una vez más al ejemplo previo, donde se describe una tabla que almacena instancias de diferentes libros, donde cada uno tiene varios atributos (o columnas), se descubre, que en una BBDD estilo NoSQL, cada registro de un libro se almacena como un documento JSON. Dentro de este documento se encontrarán diferentes elementos como el Título, Editorial, ISBN… que se almacenarán como atributos. Esta repartición de los datos inicialmente puede parecer ineficiente, pero, sin embargo, permite que la base de datos sea mucho más flexible y escalable. Sin embargo, la estructura de organización de una BBDD no está fijada, como si ocurre en SQL. Si no que existen varias. (14)

[3] **Big Data**: Conjunto de datos de gran tamaño y complejos. Normalmente analizados mediante aplicaciones de procesamiento de datos no tradicionales.

Clave y Valor

Este tipo de base de datos almacenan la información como una pareja formada por un atributo y un valor. Esta pareja debe ser única, es decir, no puede haber dos iguales, ya que el atributo será único. Este tipo de base de datos se diseñó para poder proporcionar una latencia de milisegundos para casi cualquier carga de trabajo.
No existe un esquema preestablecido para almacenar la información, sino que se permite que el usuario haga copias en diferentes formatos, permitiendo así una vez más, consultas de información eficaces y rápidas.
Un ejemplo popular en el mercado actual sería Oracle NoSQL o Amazon DynamoDB.

Documentos

Otra forma de almacenar la información es representándola como un objeto o documentos JSON. Estos facilitan la consulta de datos y comprensión, debido a la gran flexibilidad de estos, la BBDD podrá adaptarse a las necesidades una aplicación.
Suelen utilizarse en escenarios como un catálogo, perfiles de usuarios… Es decir, cuando se tiene un documento único al que se le irá añadiendo información, que evoluciona con el tiempo. Amazon DocumentDB o MongoDB son las BBDD basadas en este formato, que actualmente se utilizan con más frecuencia.

Gráficos

En el caso de querer gestionar aplicaciones con una gran carga de datos conectados, las bases de datos de tipo Gráficos será la mejor opción.
Es común encontrar un esquema similar en la gestión de redes sociales o gráficos de conocimiento. El propósito es crear y ejecutar aplicaciones donde la mayor parte de la información provine de las relaciones entre los datos. Un ejemplo actualmente popular en el mercado seria Amazon Neptune o Neo4j.

En memoria

Este tipo de bases de datos almacenan la información en memoria, en lugar de discos o SSD, como si hacen los modelos más tradicionales. La decisión nace con el objetivo de minimizar los tiempos de respuesta, eliminando todo acceso a discos. Además, también se reduce el riesgo de pérdida de datos en el reinicio de un servidor o apagado forzoso. Se almacenan los datos en *logs* o mediante *snapshots*[4].

[4] Snapshot: El estado de un sistema en un momento en concreto.

13. Estudio del rendimiento

Se quiere estudiar si existe una diferencia de rendimiento, entre la ejecución de la misma aplicación en un entorno dominado por CPU, o, por otro lado, una máquina que delega la mayor parte de los cálculos a la unidad GPU.

Antes de hacer el estudio de rendimiento, es necesario entender cómo se comunica una máquina con una base de datos.
Cuando se trata con grandes escenarios donde por ejemplo una empresa almacena la información en un centro de datos externo, la comunicación entre este y la aplicación o programa que hace las consultas, ocurre a través de la red. Se envían paquetes de información entre estos mediante los cuales se solicita y se sirven las consultas. Por supuesto, este proceso es más complejo porque a menudo cuenta con métodos de encriptación, diferentes protocolos de comunicación… Sin embargo, para este proyecto, de un tamaño mucho más reducido, se tendrá la base de datos en local. Esto implicará que la comunicación sea mucho más rápida, ya que todo ocurre en la misma máquina.

Tradicionalmente, ya sea un centro de datos o en local, la información de la base de datos se almacena en un disco. La información sobre la arquitectura de un PC que se ha desarrollado en apartados previos permitirá una mejor comprensión de como la CPU es alimentada con la información necesaria.

A continuación, puede observarse un esquema simplificado de la jerarquía de memoria que hay por encima del disco duro, es decir, entre la memoria principal y el procesador.

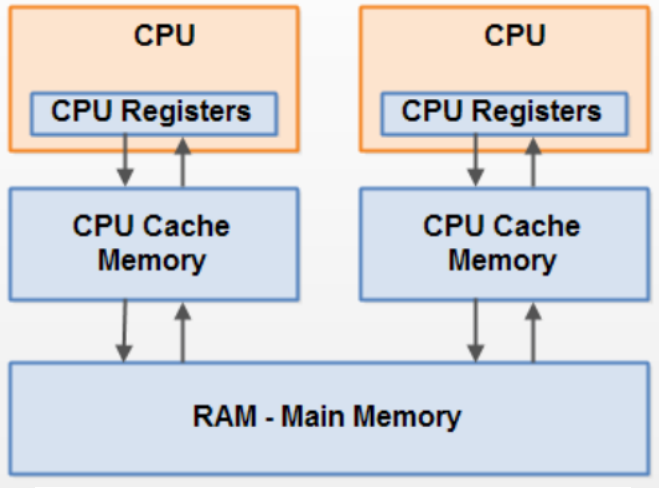

Figura 12: Comunicación entre memoria y CPU. Fuente: (18)

Los registros son el componente más pequeño y rápido capaz de almacenar datos dentro de la jerarquía.

Estos funcionan en conjunto a la CPU y guardan todo tipo de información, desde parámetros que determinan que parte del código se ejecutará, hasta la información contenida en una variable. No obstante, debido a la limitación de tamaño de datos que puede ser almacenado, cuando el procesador no encuentra la información que necesita en estos, pasará a consultar a la memoria cache.

La caché, que también puede dividirse en varios niveles, es el segundo componente más rápido en esta estructura. Guardará toda aquella información que, o bien el procesador consulta con frecuencia, o, aquellos datos que se prevén que serán utilizados en futuras consultas.

Finalmente, la RAM (*Random Access* Memory) es una memoria de corto plazo, que guarda la información necesaria para ejecutar un programa. Un ejemplo de funcionamiento sería suponer que una máquina tiene tres aplicaciones diferentes y una gran cantidad de información almacenada; todos estos datos se encuentran inicialmente en el disco duro.
Cuando se arranca una de las aplicaciones, la información relacionada a esta se almacena temporalmente en la RAM, para un acceso fácil y rápido. Cuando termine la ejecución de la aplicación y una nueva quiera ser ejecutada, la RAM pasará a guardar los nuevos datos, una vez más, de forma temporal.

El resto de información siempre podrá ser encontrada en el disco duro. Como es el elemento más alejado del procesador, será también el que ofrezca la mayor latencia, en comparación a otros módulos de memoria.

Por otro lado, la comunicación entre GPU y memoria funciona de una forma similar.

Los *Streaming Multiprocessor* se comunican con la memoria caché asignada a esa unidad. Cuando los datos solicitados no se encuentran en ese módulo, se consultará la caché compartida por todos los SM. Asimismo, en caso de querer utilizar información que no se encuentra en la caché, pasará a consultarse la memoria Gráfica, donde normalmente se encuentran los resultados de aquellos cálculos más complejos que ha ejecutado la GPU.
Por último, se accederá a la memoria principal.

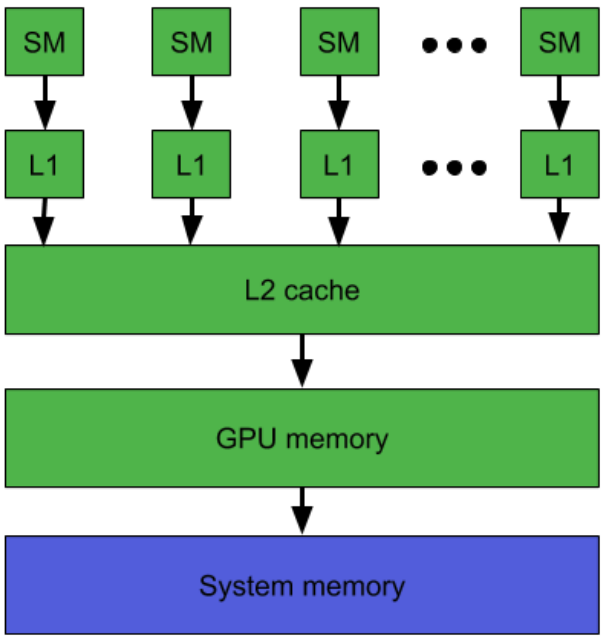

Figura 13: Comunicación entre memoria y GPU. Fuente: (19)

Los componentes más importantes que han tomado parte durante el desarrollo del estudio han sido los siguientes.

13.1. Redis

Se ha escogido Redis como la base de datos utilizada para ambos entornos. Esta se caracteriza por ser NoSQL y por almacenar toda la información en memoria RAM, de forma contraria a como hace una BBDD tradicional, que guarda los datos en disco.
Esta funcionalidad ayuda a reducir la latencia y consecuentemente ofrece un número elevado de operaciones por segundo, o, en otras palabras, un tiempo de respuesta más rápido.

Esta decisión se toma, ya que la información con la que trabajará la aplicación es de un tamaño reducido, y, por lo tanto, esta puede almacenarse fácilmente en memoria. Para futuras ampliaciones donde posiblemente se pudiera trabajar con un tamaño de datos superior, probablemente sería más adecuado cambiar a otro modelo. Sin embargo, queda mucho recorrido para alcanzar el límite y por lo tanto Redis ha sido considerada una buena opción.
Adicionalmente, se utilizará la instancia RedisJSON. Este módulo proporciona soporte JSON a la base de datos. Facilita el almacenaje, lectura y actualización de ficheros JSON en la base de datos Redis. De esta forma, se consigue una mejor integración y mejora la velocidad de acceso.
Cabe destacar que todo fichero JSON debe tener un set de clave/valor. Cada elemento almacenado tiene asignado una clave única. Esta en conjunto a un valor forman un identificador único para cada conjunto de datos, así pues, no existirán dos elementos que compartan el mismo set de clave/valor.

13.2. Docker

Un contenedor es un entorno software que contiene los componentes necesarios para poder ejecutar una aplicación. Entre estos se encuentran los ficheros de configuración, librerías, dependencias…
Este tipo de tecnología permite que el código contenido pueda hacerse correr en diferentes entornos hardware y que la adaptación a diferentes infraestructuras pase a ser una tarea más fácil.
Esta idea nace a raíz de los continuos problemas software que aparecen cuando se migra una aplicación a un nuevo entorno. Y los contenedores proponen una solución al empaquetarlo todo en una única imagen que correrá y se adaptará a los recursos de la máquina donde se acoja la aplicación.
Para el desarrollo del proyecto se utiliza Docker que es una herramienta software que permite desarrollar, empaquetar y ejecutar la imagen de una aplicación encapsulada en un container.

13.3. CUDA

La computación en GPU se basa en la habilidad de paralelizar cálculos. Cuando se marca como objetivo que sea este módulo se encargue de la ejecución de determinadas partes de un programa, es necesario adaptar el código para que se siga dicho comportamiento.

Para esta memoria, se recogerá el código fuente de la aplicación que se estudia en el trabajo, y se adaptará para la correcta ejecución tanto en CPU, es decir, de forma serializada; como en GPU, donde se ejecutará de forma paralela y se aplican los principios de programación CUDA. Cabe nombrar, que se empleará el paquete OpenSource Numba.

Se define el concepto *Kernel CUDA* como una función que se ejecuta en la GPU.
Supongamos que se genera una función llamada *cudakernel0* que trabajará con un *numpy array*. Estos funcionan como un array estándar, pero se definen como *numpy* para que la versión CUDA de Numba, interactúe de forma correcta con Python.
Además, será necesario modificar la definición de la función que usará el módulo kernel. Esto se consigue añadiendo **@cuda.jit** y las siguientes librerías:

```
from numba import cuda
import numpy as np
```

Una vez se ha adaptado el código, el siguiente paso es arrancar el kernel utilizando el siguiente comando:

```
kernelname [gridsize, blocksize] (arguments)
```

Una GPU organiza las tareas computacionales en diferentes bloques, donde cada uno de estos utiliza uno o varios threads. El primer parámetro *gridsize* determina el número de bloques, y *blocksize* definirá el número de threads por bloque. Consecuentemente, el número de total de threads puede obtenerse multiplicando *gridsize* por *blocksize*.

Sin embargo, simplemente ejecutar el kernel no asegurará que el código funcione correctamente, ya que es muy importante establecer un orden entre la ejecución de los threads.
Supongamos que la función *cudakernel0* modifica varias veces los valores del *array* a su número opuesto, es decir, si tiene valor 0 pasará a ser un 1 y viceversa. El problema yace en que el valor del mismo elemento puede variar entre una ejecución y otra del mismo código, ya que dos threads podrían leer el mismo valor a la vez y, por lo tanto, devolver el número contrario, o, por otro lado, un *thread* podría leer el valor 0 y negarlo, y a continuación el siguiente *thread* leería el nuevo valor y lo negaría, volviendo al resultado inicial.

Una solución para este caso sencillo sería utilizar la función *cuda.grid()*, que retorna la posición actual de un thread dentro del bloque, y asignar ese valor a la posición del *array* que quiere modificarse. De esta forma, se evitarían superposiciones entre los elementos.

En el caso de la aplicación, todos los datos están almacenados en un fichero JSON, el cual es recorrido elemento a elemento para obtener el set clave/valor ,mediante el cual se hace una lectura, escritura …

Añadir el kernel, cabeceras y librerías de CUDA no serían suficiente para paralelizar este código, y por lo tanto, se ha modificado el funcionamiento.
Primeramente, se recogen todos los valores del fichero y se almacenan en un *numpy array*. Este será posteriormente tratado por una función kernel. Consecuentemente la GPU podrá ejecutar varias operaciones al mismo tiempo, a diferencia de la CPU donde se haría una lectura/escritura una a una.

13.4. Gradient Notebooks

Gradient es una herramienta diseñada para acelerar procesos en la nube, trabajar con IA e incluso *machine learning*. La compañía *Paperspace* introduce los Gradient Notebooks que son una aplicación Jupyter basada en web, que permite hacer uso de GPU.
Por otro lado, Jupyter es un software libre que ofrece un servicio de computación con soporte para una gran variedad de lenguajes de programación.

Así pues, el propósito de los Notebooks es presentar un entorno de desarrollo Jupyter interactivo, que es el que será utilizado durante este proyecto.

13.5. Comunicación base de datos con computador

Para dar de alta la base de datos en local se siguieron los siguientes pasos.

Primeramente, se empieza con la instalación de Docker, en este caso para un ordenador Intel que usa el procesador 2,9 GHz Dual-Core Intel Core i5, tiene una memoria RAM de 8 GB y utiliza el sistema operativo MacOS Monterey.

Una vez se ha completado la instalación se descargará y ejecutará el container *Redis Enterprise Software Docker* con el siguiente comando:

docker run -d --cap-add sys_resource --name rp -p **8443:8443** -p 9443:9443 -p **12000:12000** redislabs/redis

El comando *docker run* crea un container con la imagen especificada, en este caso, *redislabs/redis*. Además, se abre el puerto 8443 para conexiones HTTPS y el 9443 para conexiones REST API. Por último, el puerto 1200 se abre para las conexiones cliente, es decir, la comunicación con la base de datos.

Se define un Clúster de Redis como una malla formada por varios nodos. Estos están interconectados entre sí utilizando una comunicación TCP. El *Clustering* permite que la base de datos se comparta a través de diferentes nodos.

Para esta memoria se usará únicamente un nodo, que es donde se acogerá la instancia de la base de datos, ya que para la magnitud del proyecto es suficiente. Así pues, el siguiente paso será dar de alta un clúster y configurar un nodo. Cabe destacar que tanto Redis como Docker ofrecen modalidades de pago, pero no han sido necesarias para la configuración que se ha empleado.

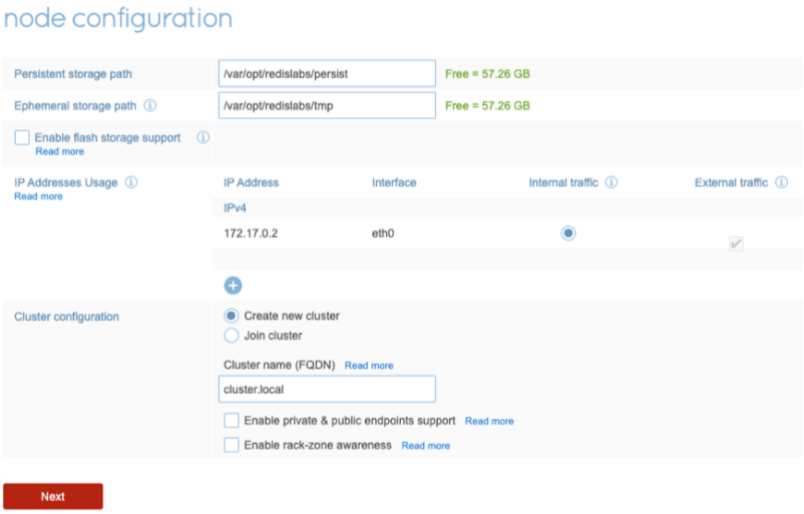

Figura 14: Configuración nodo Redis. Fuente: Elaboración propia

Una vez se ha configurado el entorno donde crear la base de datos , el siguiente paso será la creación de esta. Para la configuración, cabe destacar que estará en memoria RAM, y se le asigna una capacidad de 1 GB. Además, se configura el *endpoint* en el puerto 12000.

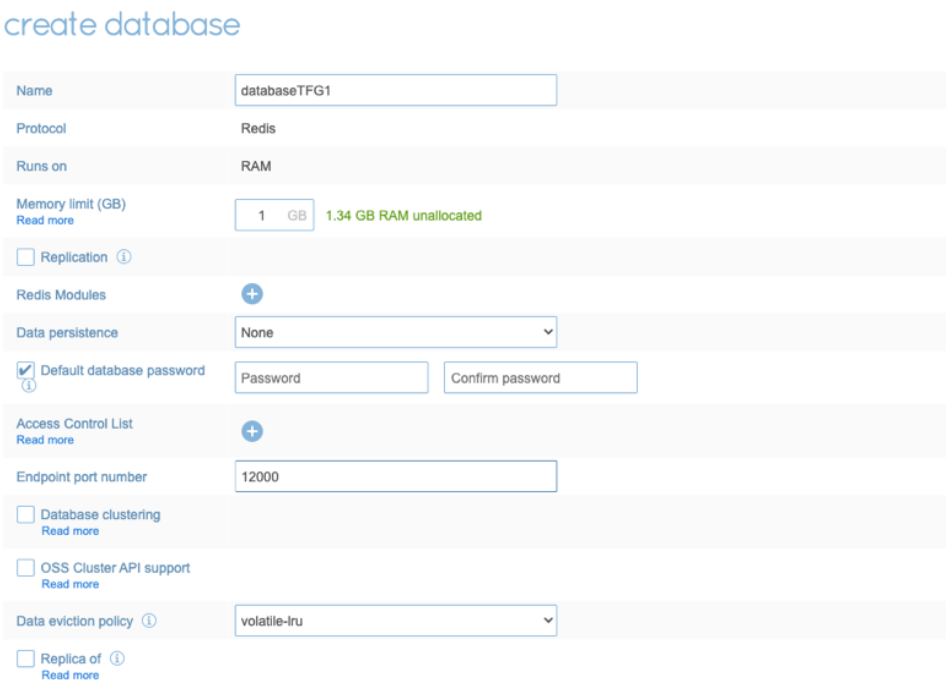

Figura 15: Configuración BBDD Redis. Fuente: Elaboración propia

Finalmente, el último paso restante es conectarse a la base de datos. La conexión se lleva a cabo mediante la línea de comandos *redis-cli*, que permite la interacción con la BBDD. Una vez se confirma el correcto funcionamiento puede empezar a testearse la aplicación en cuestión.

Originalmente, se desarrollaron una serie de comparaciones, explicadas en los siguientes apartados, que tuvieron lugar en el ordenador Intel descrito en el comienzo de esta sección. El objetivo era recoger los datos en la plataforma CPU y a continuación migrar este proceso al Clúster proporcionado por la plataforma DAC. Por motivos de incompatibilidad (permisos, almacenamiento limitado…) finalmente se decidió utilizar el entorno Gradient para acoger el container y llevar a cabo la comparación entre plataformas.
La máquina elegida cuenta con 8 CPU Intel Xeon E5-2623 v4 2.60Ghz, 32.2 GB de memoria RAM y una GPU Quadro M4000 de 8 GB.

El proceso de migración implicó subir el container a la nube de la plataforma Docker, mediante el comando *docker push*. Seguidamente, se dio de alta un Notebook y se configuró el acceso al container ya existente.

Inicialmente, se planteó usar el siguiente código para poder paralelizar la escritura en la base de datos.

```
from numba import cuda
import numpy as np

# Funcion a ejecutar en GPU
@cuda.jit
def cudakernel_Insert(array_primary_key, array_data, output):

    # localizar posicion del thread
    thread_position = cuda.grid(1)
    output[thread_position] = client.jsonset (array_primary_key[thread_position], array_data[thread_position],
array_primary_key[thread_position])

    return (output[thread_position])
# Proceso
cudakernel_Insert [1, 1555] (primary_key, data, output)
```

Una vez se ha transformado el fichero JSON en un array con toda la información y las *Primary Keys,* este es enviado a la función Kernel CUDA llamada *cudakernel_Inser*t para que puedan paralelizarse las escrituras.

En el *array output* se almacena el resultado de cada operación, es decir, si la escritura se ha completado con éxito o no. Para evitar que más de un thread trabaje con el mismo elemento, se emplea la función *cuda.grid()* que forzará a que cada uno se encargue de una posición diferente del array. Sin embargo, se determinó que no era posible hacer una llamada a una función externa desde la función kernel, en este caso, a la base de datos Redis.

La decisión final fue no aplicar los cambios que permiten la paralelización con CUDA, y, en cambio, usar la funcionalidad de docker que deriva todos los recursos de la GPU al container especificado. Por lo tanto, se utilizaría el mismo código para ambas arquitecturas, pero el comando para ejecutar el docker varía. A continuación, se muestran la diferencia entre el comando para CPU y GPU respectivamente.

docker run -p 6379:6379 redislabs/redis
docker run -p 6379:6379 **--gpus all** redislabs/redis

Cabe destacar, que es necesario instalar la herramienta *nvidia-container-toolkit*.

13.6. Tipo de datos

La aplicación que va a testearse almacena un listado de 1.555 Organizaciones No Gubernamentales (ONG). Para cada uno de los elementos se creará un fichero JSON con la información correspondiente a esta, que almacenará 16 parámetros diferentes. La totalidad de todos estos datos suponen un tamaño de 14 MB.

Un fichero JSON acepta únicamente un número limitado de tipos de datos, entre los que se encuentran String, Número, Objeto, Array…
Para este caso se han utilizado nueve campos de tipo String, es decir, una cadena de caracteres. Cinco variables de tipo Array, una lista de elementos, en este caso de tipo String. Y finalmente, dos variables numéricas que siempre serán un número entero.

Durante el desarrollo del proyecto, se analiza y compara como las diferentes plataformas, ejecutan diferentes scripts que harán instrucciones de escritura, lectura y modificaciones.

Por cada elemento que se encuentra en la BBDD, o, en otras palabras, por cada fichero JSON, hay una *Primary Key*, que viene representada por un conjunto de clave/valor único. Cuando se quiere insertar un nuevo fichero JSON a la BBDD, esta comprobará que no exista ningún elemento que comparta ese identificador; esto implicará hacer una lectura de las claves que ya han sido dadas de alta. En caso de que esta sea única, se procederá a ejecutar una instrucción de escritura.

Finalmente, se analizará también como el entorno responde a modificaciones en los datos ya almacenados. Se ejecutarán instrucciones para eliminar datos y se estudiará como se responde a estas.

13.7. Inserción de datos

La función **jsonset** inserta un JSON en la base de datos, estableciendo una clave y los valores correspondientes. Para el caso de esta aplicación, la clave se define como el título de la ONG. Y, por otro lado, el valor como el *path r*oot. Ambas serán variables de tipo String.

client.jsonset (elem['title'], Path.rootPath(), elem)

A continuación, se muestra la estructura del programa para realizar las escrituras en la BBDD. Este se describe en seudocódigo para facilitar la comprensión.

```
# Carga de librerias Redis y JSON
import json
from redis import Redis
from rejson import Client, Path

# Conexión a BBDD
client = Client(host='localhost', port=12000, decode_responses=True)

if (conexión establecida):
    lectura fichero json ()
    for (items in json):
        escritura en BBDD () # client.jsonset()
else:
    print ('error al establecer conexión a BBDD')
```

El primer paso es establecer la conexión a la base de datos Redis. Esto se consigue haciendo la llamada Client y utilizando el puerto 12000, como se ha establecido previamente en la configuración de esta.

Seguidamente, se hace una lectura completa del fichero JSON y se iterará en un bucle por cada elemento definido. Es entonces, cuando se hace la escritura usando la función client.jsonset.

En la siguiente figura puede observarse como se han ejecutado 1555 escrituras en la base de datos. Esta primera prueba no ha sido paralelizada, sino que se ha ejecutado una operación después de la otra, es decir, de forma serializada.

El tiempo total de ejecución para todas las escrituras en la CPU se sitúa sobre los 23 segundos.

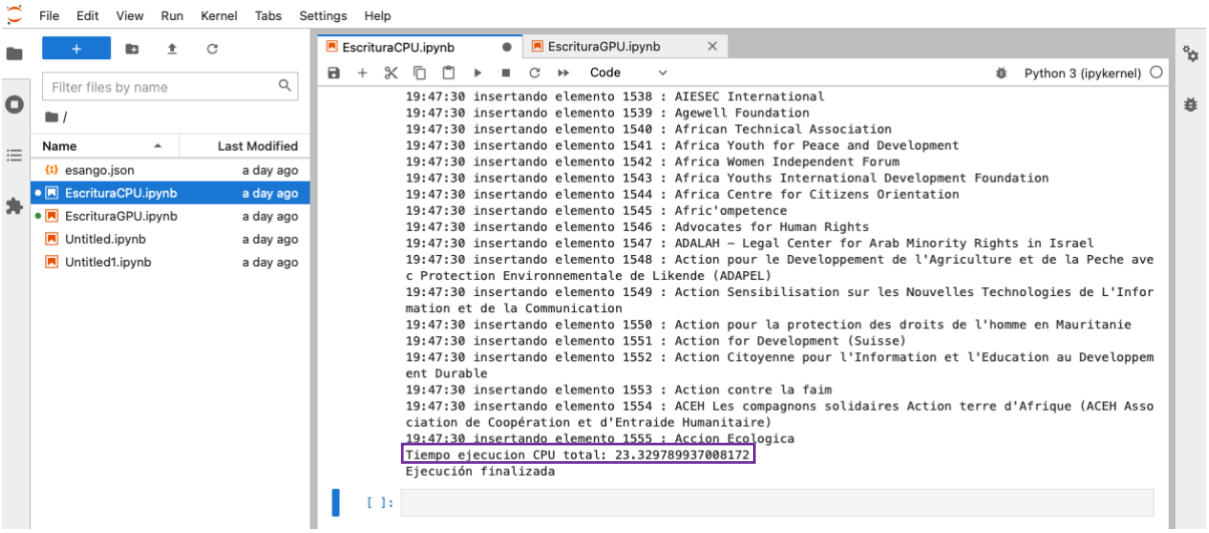

Figura 16: Escritura CPU en JupyterLab. Fuente: Elaboración propia

Además, se han recogido las métricas que reflejan el uso de la CPU durante la ejecución del programa, así como, el uso de memoria que se ha hecho.

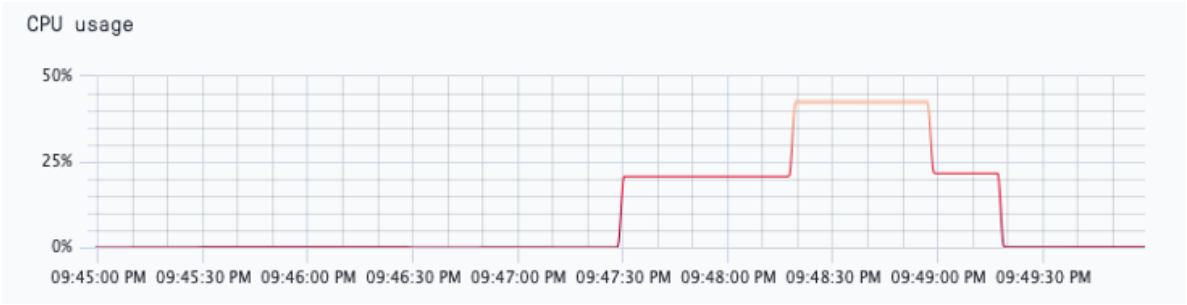

Figura 17: Gráfica del uso de la CPU durante las escrituras. Fuente: Elaboración propia

Se espera que este valor se vea reducido en la comparación con GPU, ya que esta última destaca por su capacidad de paralelizar, y, por lo tanto, puede hacer un mayor número de cálculos en un tiempo inferior. Para esta primera prueba, se tomará el valor máximo de 42.50% de uso del procesador.

Por otro lado, se repite el mismo experimento, pero ahora asignando todos los recursos de la GPU disponibles al container. El tiempo total de ejecución utilizando los recursos GPU es de aproximadamente 5.02 segundos.

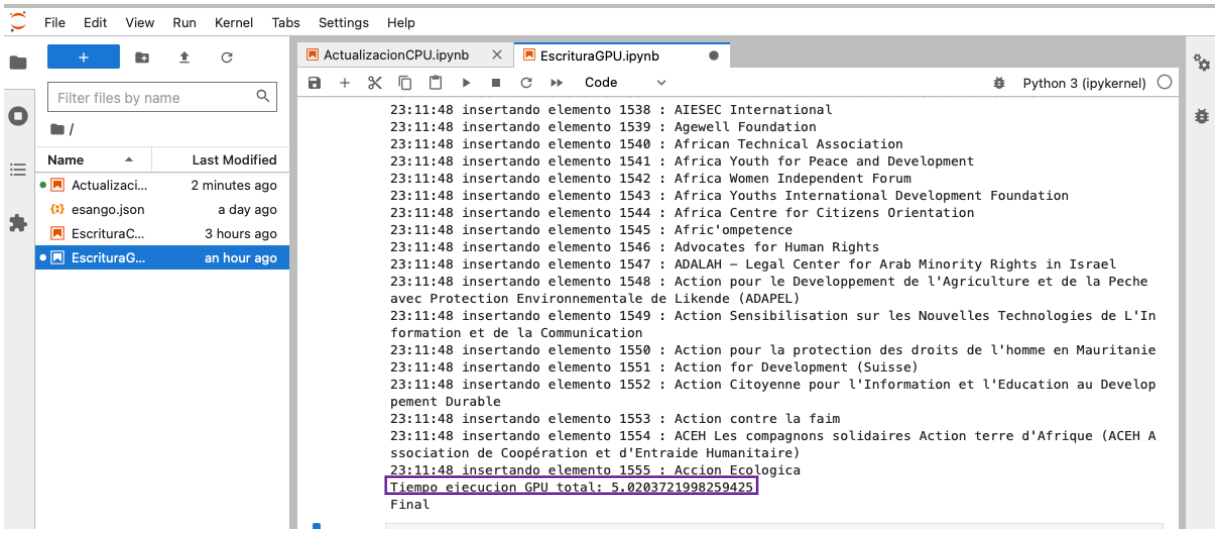

Figura 18: Escritura GPU en JupyterLab. Fuente: Elaboración propia

La ejecución del mismo número de escrituras ha supuesto un 16.6% , en el pico más alto, de los recursos de la CPU, y únicamente un 6% de los recursos GPU.

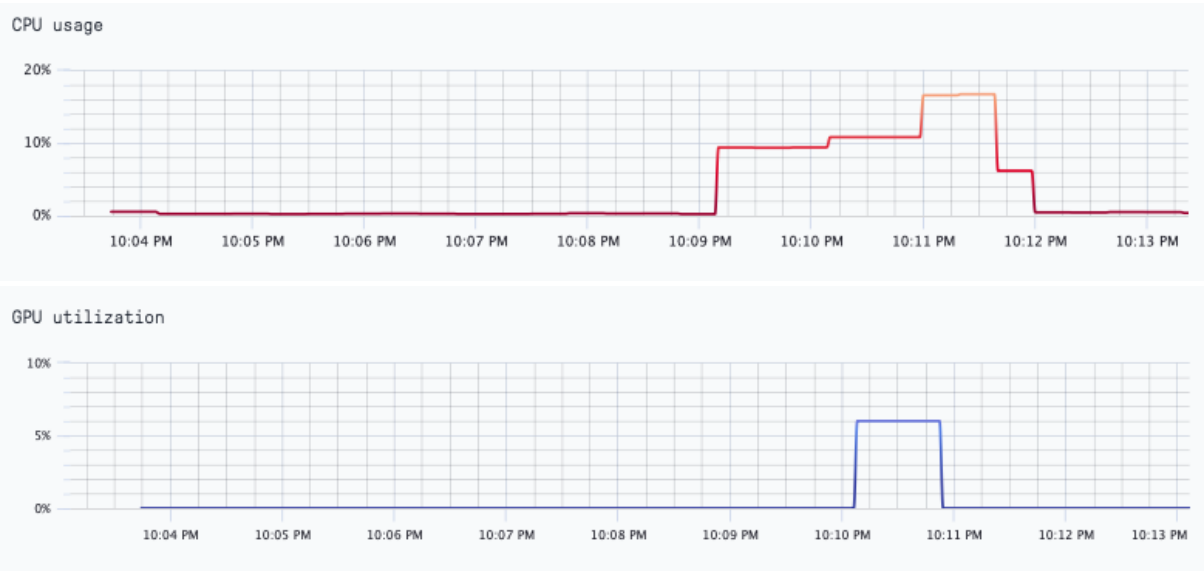

Figura 19: Gráfica del uso de la GPU y CPU durante las escrituras. Fuente: Elaboración propia

Además, se han recogido otras métricas como el consumo energético durante el periodo de la ejecución o la temperatura que ha alcanzado el procesador.

El primer parámetro alcanza un valor de 45 W que se mantiene fijado durante la ejecución, pero inicialmente este se situaba en 11 W, por lo tanto, se ha experimentado un aumento de 34 W.

La segunda variable refleja una temperatura máxima de 28º, sin embargo, esta también se encontraba con un valor superior a 0 inicialmente, en este caso alrededor de 25/26º.

Así pues, se ha decidido no tener en cuenta esta métrica para las siguientes mediciones, debido a la poca diferencia.

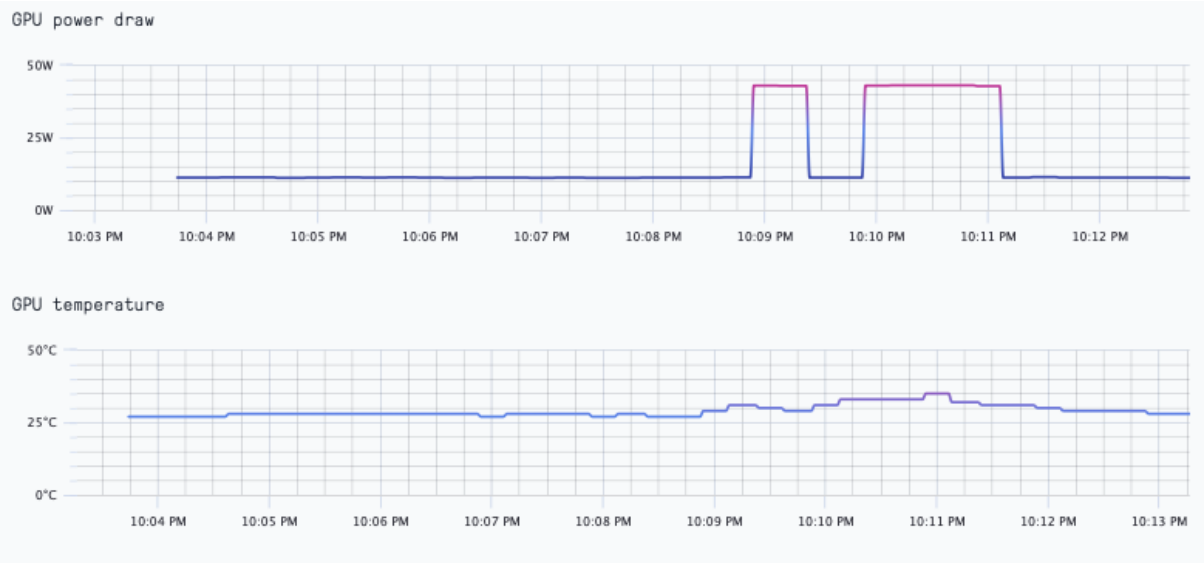

Figura 20: Gráfica del consumo energético y de temperatura para una GPU durante las escrituras. Fuente: Elaboración propia

13.8. Obtención de datos

La función **jsonget** hace una búsqueda entre todos los datos almacenados en la BBDD y retorna el valor correspondiente para la *Primary Key* que se ha definido en el parámetro clave. A continuación, se define la sintaxis de la función, así como el código desarrollado para la lectura.

client.jsonget (elem['title'])

```
# Carga de librerias Redis y JSON
import json
from redis import Redis
from rejson import Client, Path

# Conexión a BBDD
client = Client(host='localhost', port=12000, decode_responses=True)
if (conexión establecida):
    lectura fichero json ()
    for (items in json):
        búsqueda en BBDD # client.jsonget()
else:
    print ('error al establecer conexión a BBDD')
```

Nuevamente, se establece la conexión a la base de datos Redis.
Para poder medir el rendimiento y consumo de esta función, se ha supuesto el escenario donde se desea hacer una lectura de todos y cada uno de los elementos. Esto se consigue aprovechando el contenido del fichero JSON para seleccionar diferentes claves en el bucle.
La imagen que aparece a continuación muestra como la BBDD sirve elementos almacenados e imprime la clave asignada a cada uno de estos. Confirmando que se ha podido encontrar los datos de forma exitosa.

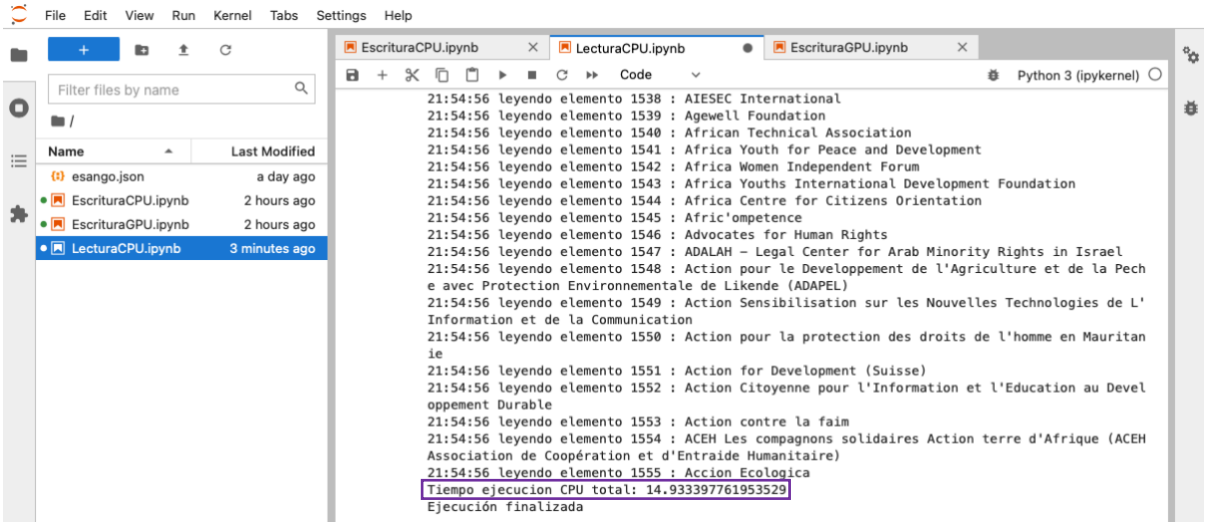

Figura 21: Lectura CPU en JupyterLab. Fuente: Elaboración propia

La ejecución de la lectura de todos los elementos almacenados en la BBDD, han supuesto un tiempo de 14.93 segundos.

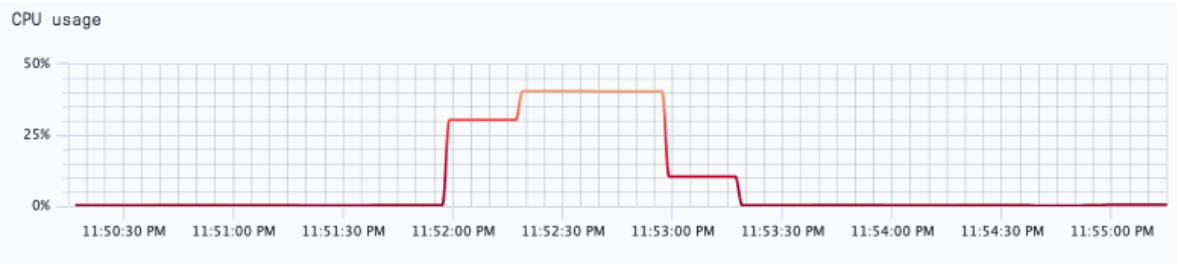

Figura 22: Gráfica del uso de la CPU durante las lecturas. Fuente: Elaboración propia

El uso máximo que se ha hecho del procesador ha sido de 40.4 %, bastante similar al utilizado previamente.
Una de las primeras diferencias que salta a la vista al comparar estos datos con los recogidos en el experimento previo, es la diferencia en el tiempo de ejecución. Donde para las escrituras se fija en 23 segundos y para las lecturas en aproximadamente 15 segundos.

La diferencia entre ambos valores se otorga a la existencia del conjunto *key/value*.

La BBDD busca una clave que coincida con el valor seleccionado, una vez esta ha sido encontrada, la información puede ser servida inmediatamente ya que se sabe con certeza que no habrá dos ONG que compartan la misma clave. Esta condición se garantiza durante la escritura, ya que al insertar un nuevo dato deberá hacerse una lectura previa de las *Primary Key,* para la gestión de duplicados. Por lo tanto, por muy rápido que sea este proceso o por mayor paralelización que se implemente, siempre será más lento que una simple lectura.

Para la segunda parte de este escenario, se repite la ejecución utilizando una vez más todos los recursos de la GPU, es decir, paralelizando las operaciones.

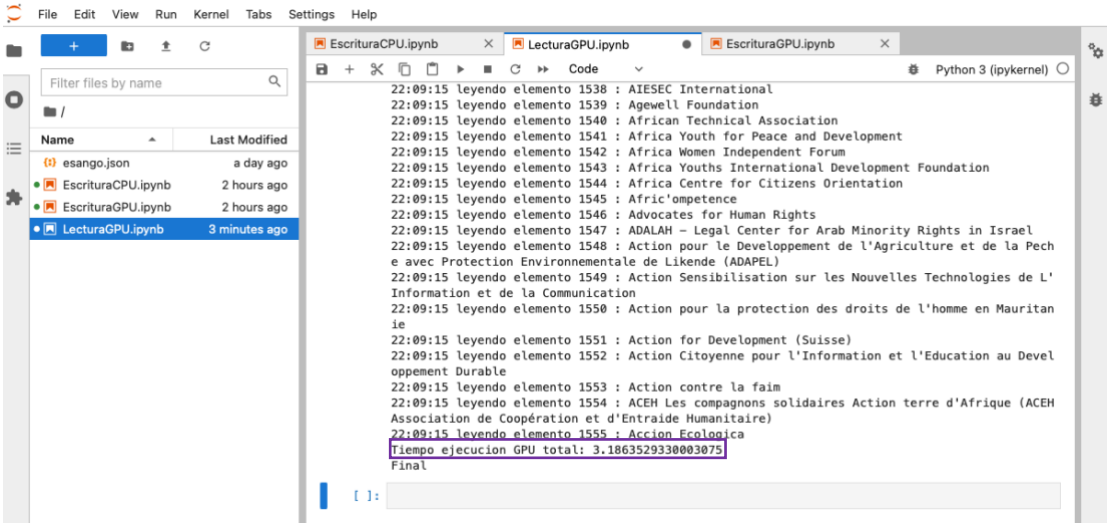

Figura 23: Lectura GPU en JupyterLab. Fuente: Elaboración propia

Se ha observado un tiempo de ejecución total, para hacer una lectura completa de la BBDD de 3.18 segundos. Los valores obtenidos para este experimento reflejan que el porcentaje máximo de CPU empleada ha sido alrededor de un 16%, y se ha dado un uso del 4% de la GPU.

Figura 24: Gráfica del uso de la GPU y CPU durante las lecturas. Fuente: Elaboración propia

Cabe destacar, que se ha observado un consumo energético de 43.5 W durante la ejecución. Sin embargo, el consumo inicial se situaba en 11.5 W, así pues, se ha experimentado un incremento de 32 W.

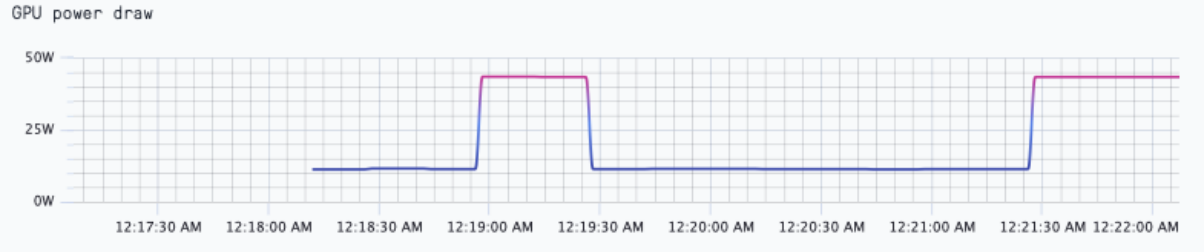

Figura 25: Gráfica del consumo energético para una GPU durante las lecturas. Fuente: Elaboración propia

13.9. Modificación de datos

Para el último experimento se ha decidido actualizar la información de la BBDD. La función **jsonset** usada para la inserción, es la misma que se utilizará para modificarlos.

La diferencia se encuentra en que la *Primary Key*, mediante la cual se identifica la ONG en la base de datos, ya estará dada de alta, y, por lo tanto, se reescribirá la información que haya sido especificada en la función.

```
#carga de librerias Redis y JSON
import json
from redis import Redis
from rejson import Client, Path

#conexión a BBDD
client = Client(host='localhost', port=12000, decode_responses=True)

if (client): # conectado
    leer_fichero_json()
    for (items in json):
        actualización en BBDD
else:
    print ('error al establecer conexión a BBDD')
```

Para el desarrollo de esta prueba se consideran únicamente valores válidos, es decir, en el caso de la escritura se insertarán únicamente valores nuevos, y para la actualización se modificarán aquellas ONG ya dadas de alta.
El código desarrollado confirma la conexión con la BBDD y posteriormente ejecuta la actualización de todos los elementos. En el apartado anterior, donde se explica el funcionamiento de la obtención de datos, se menciona que una escritura consta de dos partes, que son la lectura previa de las claves y seguidamente la escritura. Para este caso, se ha decidido actualizar todos los campos de cado uno de los elementos. De esta forma, observando las diferencias entre la escritura y actualización, podrá verse que proporción de recursos o tiempo se dedica a generar una nueva entrada.

El tiempo final dedicado a la actualización ha sido de aproximadamente 20.4 segundos.

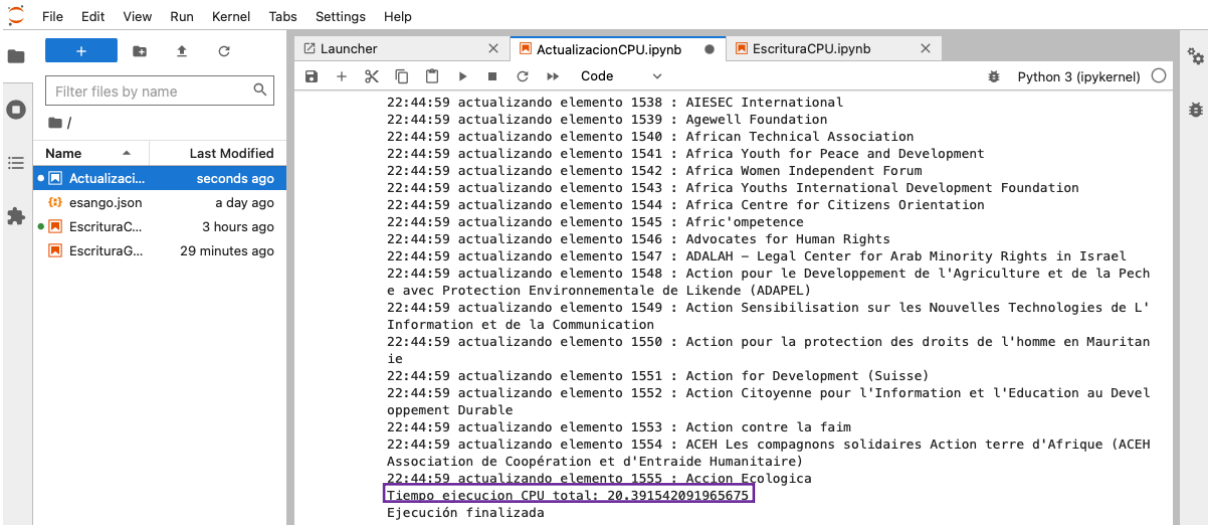

Figura 26: Actualización CPU en JupyterLab. Fuente: Elaboración propia

Por otro lado, el consumo de la CPU se mantiene prácticamente estable en un 35% durante la ejecución.

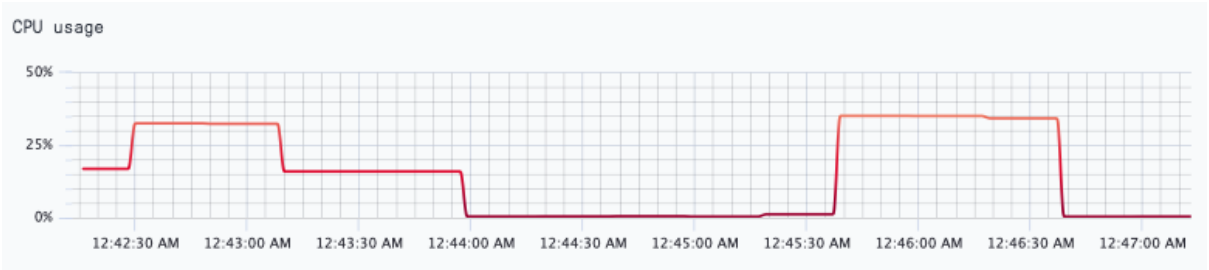

Figura 27: Gráfica del uso de la CPU en la actualización. Fuente: Elaboración propia

Finalmente, repetir esta prueba en un entorno GPU tendría las siguientes implicaciones.

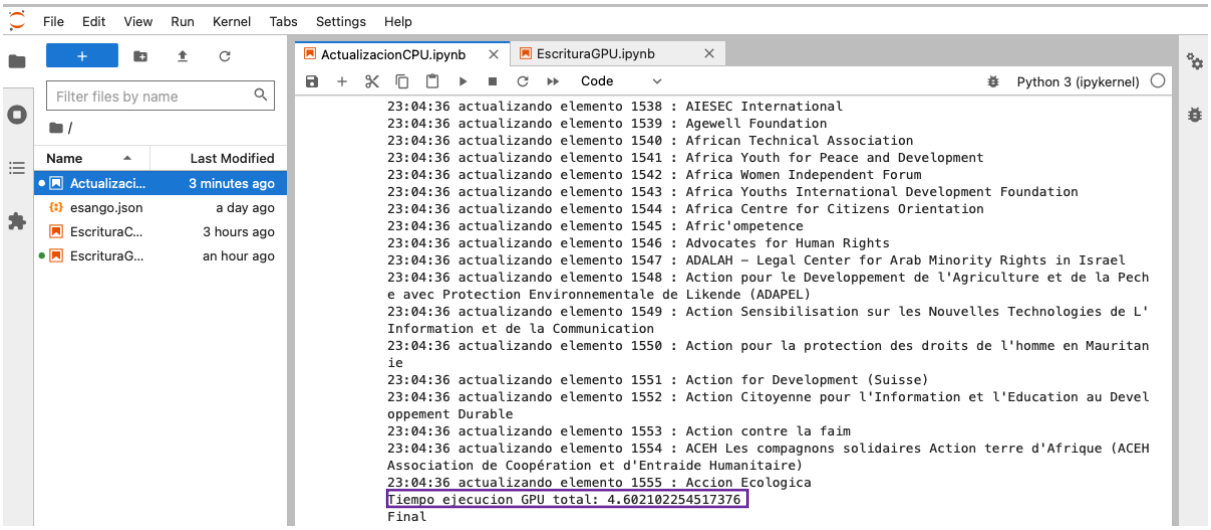

Figura 28: Actualización GPU en JupyterLab. Fuente: Elaboración propia

Se consigue reducir el tiempo de ejecución hasta 4,6 segundos aproximadamente. Cuando se compara este valor con el obtenido en la escritura de datos, se observa una diferencia de alrededor de medio segundo.

El cambio que se observa entre los dos valores se puede otorgar a que en el segundo experimento no ha sido necesario dar de alta un nuevo elemento, simplemente reescribir los campos.

La utilización que se le ha dado a la CPU ronda alrededor de un 22%, mientras que la GPU se mantiene en un 4%.

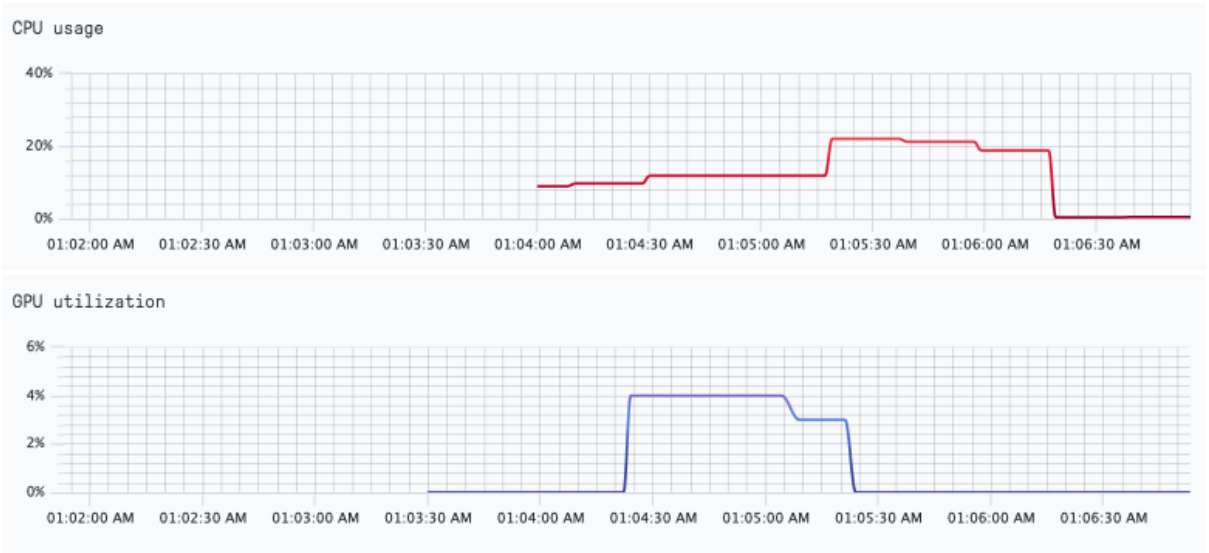

Figura 29: Gráfica del uso de la GPU y CPU durante las actualizaciones. Fuente: Elaboración propia

Además, el consumo energético durante este programa se sitúa en 44 W. Teniendo en cuenta el valor inicial, se observa un incremento de 32 W.

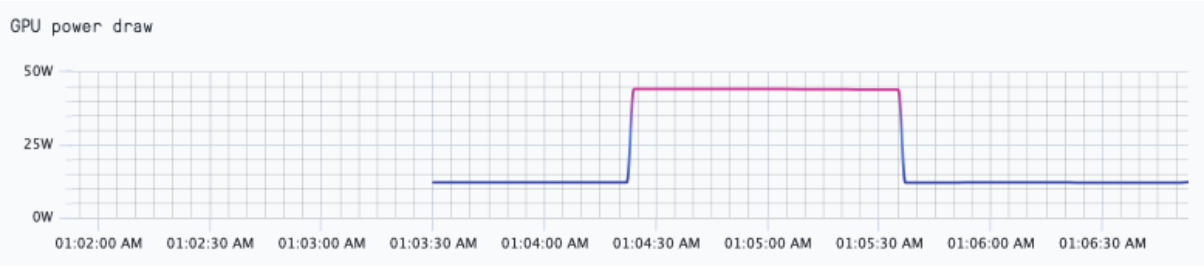

Figura 30: Gráfica del consumo energético para actualización en GPU. Fuente: Elaboración propia

13.10. Limpieza de la base de datos

Finalmente, la función **client.flushall ()** se encarga de vaciar toda la información registrada en la BBDD. Esta elimina todas las claves primarias.
Este programa se ha ejecutado con frecuencia durante las pruebas, para poder hacer un limpiado de los datos y asegurar que las métricas recogidas no estaban afectadas por pruebas anteriores.

```
from datetime import datetime
from timeit import default_timer as timer

# Conexión a BBDD
client = Client(host='localhost', port=12000, decode_responses=True)
if (conexión establecida):
    Vaciar la BBDD # client.flushall()
```

Cabe destacar, que para todos y cada uno de los experimentos se ha proporcionado la información recogida en una de las varias ejecuciones que se han llevado a cabo. No obstante, se ha intentado iniciar las pruebas justo después de reiniciar la máquina o después de asegurarse que ningún otro proceso estaba trabajando. Sin embargo, resulta prácticamente imposible que el mismo programa sea ejecutado exactamente en el mismo tiempo, así pues, el breve razonamiento que acompaña a la descripción de las pruebas puede estar sujeto a cambios, pues estos valores pueden variar.

A raíz de estas diferencias, en el siguiente apartado se lleva a cabo un análisis para estudiar las diferencias entre CPU y GPU. A pesar de haber generado múltiples ejecuciones para cada experimento, siempre ha habido una gran mejora al cambiar de plataforma.

14. Análisis de los resultados

Para medir la eficiencia de ambos escenarios se compararán los siguientes parámetros.

Primeramente, el tiempo de ejecución de las diferentes operaciones ayudará a valorar el rendimiento de cada uno de estos. Por otro lado, el consumo energético determinará la proporción de energía consumida por recurso utilizado. Cabe destacar, que no se tendrá en cuenta la energía disipada por el calor de los procesadores, ya que el cambio en temperatura observado ha sido mínimo. De esta forma se hará un balance sobre si la velocidad ganada compensa la energía que se ha necesitado, o, por otro lado, si es mejor alargar el tiempo de ejecución y consecuentemente reducir el consumo.
Por último, otros recursos usados, como por ejemplo la memoria, no serán valorados ya que ha sido el mismo en ambos escenarios y no supondría ningún cambio en la comparación.

Cabe destacar, que con la plataforma Gradient no han podido obtenerse las métricas de consumo energético para la CPU. Pero, sin embargo, estas han podido ser estimadas.

Se establece que el consumo genérico para un ordenador que usa tanto CPU como GPU, se requiere entre 55 y 150 W para alimentar a la CPU. Y por el otro lado, la GPU varía entre 25-350 W.
Se ha hecho una estimación utilizando una herramienta online[5], a partir de la cual se ha podido determinar que para el modelo de procesador Intel Xeon E5-2623 se requiere un total de 113 W cuando está funcionando a máximo rendimiento, es decir, en el 100% de sus capacidades. Durante las pruebas se ha podido obtener el porcentaje de utilización del procesador para cada una de ellas, así que han podido obtenerse los valores para los diferentes usos con una simple conversión.

Los datos recogidos se encuentran en la siguiente tabla.

Tabla 13: Comparación resultados CPU vs GPU. Fuente: Elaboración propia

	CPU		**GPU**	
	Tiempo (s)	Consumo (W)	Tiempo (s)	Consumo (W)
Inserción de datos	23,33	0,24	5,02	34,0
Obtención de datos	14,93	0,15	3,19	32,0
Modificación de datos	20,39	0,18	4,60	32,0
Total	**58,65**	**0,57**	**12,81**	**98**

[5] *Power Supply Calculator* ha sido la herramienta de estimación utilizada para obtener los datos.

En la columna Tiempo, se encuentra la duración total de la ejecución del programa, medida en segundos.
Por otro lado, la columna Consumo representa el gasto energético que ha generado el procesador por el tiempo ejecutado. Se ha ignorado el consumo que el componente necesita en estado inactivo, y se ha ajustado al tiempo de ejecución del programa.

Puede observarse como hay una gran diferencia entre el tiempo total de ejecución, donde se observa una mejora de hasta 4 veces superior. Pero, sin embargo, el consumo va desde 140 hasta 213 veces mayor.

A continuación, puede observarse una tabla donde se detallan las mejoras entre tiempo de ejecución total y el consumo.

Tabla 14: Segunda comparación entre los resultados de CPU y GPU. Fuente: Elaboración propia

	Tiempo	**Consumo**
Inserción de datos	4,64	141,6
Obtención de datos	4,68	213,33
Modificación de datos	4,43	177,78

Esta diferencia confirma que, por ejemplo, en el caso de la escritura, esta es hasta 4,64 veces más rápida y consecuentemente, ejecuta un mayor número de operaciones por segundo. Pero esta mejoría en la rapidez conlleva un consumo mucho más elevado, de hasta 141 W extras.

Tomando todos los valores totales en consideración, se calculan los siguientes parámetros.

Ganancia de tiempo = tiempo total CPU – tiempo total GPU = **45,84 segundos**
Diferencia en el consumo = consumo total GPU – consumo total CPU = **97,43 W**

A partir de estas variables, se concluye que, para obtener una mejora de aproximadamente 45 segundos, se necesitaría un consumo extra de unos 97 W. Esto significaría que deberían ejecutarse todos los escenarios en CPU 170 veces para alcanzar ese consumo.

15. Conclusiones

Con relación al estudio realizado durante el proyecto, sin duda puede concluirse que el uso de GPU mejora el rendimiento de las operaciones de una base de datos. Pero desde un punto de vista energético, el uso de estos componentes no siempre será conveniente.

Por un lado, se ha visto que en las GPU pueden incorporase un mayor número de núcleos en comparación a una CPU. Hay que tener en cuenta que los *Cores* de esta última, ofrecen una mayor potencia, pero cuando los de una GPU trabajan en conjunto (a pesar de ser de un tamaño inferior y ofrecer menos rendimiento) consiguen ofrecen resultados iguales o incluso superiores. Además, esta misma característica permite que cada uno de ellos consuma una menor cantidad de energía.
No obstante, el consumo total de un procesador gráfico siempre será mayor que el de una CPU, debido a la diferencia en el número total de núcleos.
Durante los últimos años se ha incorporado la utilización de GPU para diferentes tareas, y no solo para complementar al procesador central. Es por esa razón, que la gran potencia de estos componentes requiere de un consumo energético muy elevado.
Una GPU completará la ejecución de un programa en un tiempo reducido, pero con un consumo elevado. Y una CPU tendrá un consumo inferior, pero durante un tiempo prolongado.

Se plantea en el inicio del proyecto hacer una comparación entre ambos escenarios y determinar la eficiencia energética de estos.
Es cierto que designar los cálculos a una GPU puede reducir el tiempo de ejecución desde 20 hasta a 4 segundos, lo cual es un avance increíble. Sin embargo, para un proyecto de pequeño alcance como lo es este, donde se trabaja con una base de datos muy reducida, la gran diferencia de consumo no sería suficiente razón para trasladar la aplicación a este entorno. A no ser, que se valore la velocidad por encima de la eficiencia energética.

Este proyecto tiene como objetivo estudiar la adaptación de las operaciones de una base de datos, a un entorno GPU, y comparar los resultados en una arquitectura CPU.
Inicialmente, se aplica la computación basada en los Kernel CUDA para poder paralelizar el programa que testea la escritura/lectura… Sin embargo, no ha sido posible realizar llamadas a la BBDD desde esta función, y, por lo tanto, se ha trabajado en encontrar una alternativa.

Los containers de Docker han sido la solución, ya que estos pueden configurarse para que trabajen con los recursos de una GPU. Esta opción resulta incluso mejor debido a varios factores.

Por un lado, no es necesario modificar el código cliente y adaptarlo a la paralelización y utilización de *threads*. Esto es una gran ventaja, no solo porque conseguir una alta eficiencia en la adaptación de código, es una tarea compleja, sino porque, además, para cada cambio en el programa, se debería volver a hacer una revisión de este.
Por otra parte, mantener el programa libre de modificaciones y almacenarlo en un container, permite que este pueda ser testeado en otros entornos fácilmente.

Este concepto se refleja en los requisitos no funcionales y riesgos, que se introducen en el alcance del proyecto. Ya que, al utilizar un container, mejora exponencialmente la escalabilidad de la aplicación. Y no solo eso, sino que, además, se reducen las complicaciones causadas por la falta de integración en la plataforma hardware o el deterioro de esta misma.

15.1. Nivel personal

Trabajar en este proyecto me ha permitido completar parte de los conocimientos adquiridos durante el grado universitario, así como poner en práctica conceptos que conocía únicamente de forma teórica.
Además, no estaba familiarizado con la gran variedad de usos que puede tener una GPU, ya que la función más habitual de estas es mejorar los gráficos para un computador que trabaja en edición de fotos, trabajar con videojuegos…
También me gustaría destacar que me ha resultado muy interesante ver la gran diferencia en cuanto a tiempo de ejecución entre ambos escenarios.

Por otro lado, he podido trabajar con conceptos como *containers*, *notebooks, BBDD*… y ponerlos en práctica con Docker y otras herramientas. Ha sido esta parte del trabajo, la que ha resultado el mayor reto personal.
Esto se debe a que continuamente encontraba entornos que no ofrecían los recursos que necesitaba o incluso requerían de licencias profesionales. Finalmente, me alegra poder haberme familiarizado con dichas herramientas y haber sido capaz de llevar a cabo la comparación.

No obstante, uno de los mayores aprendizajes de este proyecto ha sido la organización. He aprendido la importancia de saber dividir un proyecto en pequeñas etapas, pero sobre todo, la importancia de saber mantener los plazos, lo cual ha sido una dificultad durante el progreso del trabajo.

15.2. Futuras ampliaciones

De cara al futuro, la ampliación con más potencial sería utilizar el System-On-Module (SoM) NVIDIA Jetson Nano. Este componente permitiría ejecutar Redis dentro de un container Docker con un consumo muy bajo. Durante el desarrollo de esta memoria no fue posible adquirir uno de estos kits, pero sería una buena alternativa a la plataforma Gradient.

Otra posible ampliación sería trabajar con una base de datos a tiempo real, es decir, datos que se recogen constantemente. Podría ser muy interesante ver como una GPU trabajaría en esta situación.
Las *GPU-aware database* se estudiaron al inicio del proyecto, pero fueron descartadas debido a la incompatibilidad con las pruebas diseñadas. Además, estas son comúnmente usadas para trabajar con información a tiempo real, debido a la alta velocidad de procesado, y no era el caso con el que se trabajaba.

Finalmente, podría ampliarse el estudio para trabajar con otro tipo de parámetros. En este caso se han realizado las pruebas con *Strings* sobre la inserción, lectura y actualización de datos.

16. Bibliografía

1. **Iglesias, Angel Luis Sanchez.** La arquitectura de un procesador: Diseño de los elementos de un procesador. *About Español.* [En línea] 1 de 11 de 2019. https://www.aboutespanol.com/que-es-la-arquitectura-de-un-procesador-841131.
2. **Thornton, Scott.** Microcontroller Tips. *Microcontroller Tips. RISC vs. CISC Architectures: Which one is better?* [En línea] 9 de 1 de 2018. https://www.microcontrollertips.com/risc-vs-cisc-architectures-one-better/.
3. **Fernández, Yúbal.** Xataka. [En línea] Xataka, 16 de 4 de 2021. https://www.xataka.com/basics/que-significa-que-mi-cpu-sea-de-32-o-64-bits-y-cual-es-la-diferencia.
4. **Future Learn.** Future Learn. *Architecture of a CPU.* [En línea] https://www.futurelearn.com/info/courses/how-computers-work/0/steps/49283.
5. **López, Javier.** Seguro que sabes lo que es una tarjeta gráfica, pero ¿qué es una GPU? *Hardzone.* [En línea] Hardzone, 14 de 6 de 2021. https://hardzone.es/reportajes/que-es/gpu-caracteristicas-especificaciones/.
6. **Meselhi, Mohamed A., y otros.** Fast Differential Evolution for Big Optimization. *ResearchGate.* [En línea] 12 de 2017. https://www.researchgate.net/figure/CPU-vs-GPU-architecture-each-blue-square-represents-one-core_fig1_323281068.
7. **Gao, Hao.** Medium. *Basic Concepts in GPU Computing.* [En línea] 10 de 10 de 2017. https://medium.com/@smallfishbigsea/basic-concepts-in-gpu-computing-3388710e9239.
8. **Levinas, Mantas.** Cherry Servers. *What Is GPU Computing And How Is It Applied Today?* [En línea] 16 de 11 de 2020. https://www.cherryservers.com/blog/what-is-gpu-computing.
9. **Alonso, Rodrigo.** Hardzone. *La Ley de Moore, explicada para que la entiendas.* [En línea] 30 de 6 de 2021. https://hardzone.es/reportajes/que-es/ley-de-moore/.
10. **Castillo, José Antonio.** Profesional Review. *Nanómetros: Qué son y en qué afectan en nuestra CPU.* [En línea] 9 de 10 de 2019. https://www.profesionalreview.com/2019/10/09/nanometros-que-son/.
11. **Scientific Computing World.** GPUs aren't going to replace CPUs, but they are here to stay. *Scientific Computing World.* [En línea] 7 de 11 de 2017. https://www.scientific-computing.com/analysis-opinion/gpus-arent-going-replace-cpus-they-are-here-stay.
12. **Ayuda Ley.** Ayuda Ley y Proteccion Datos. *Bases de datos relacional ¿Qué es y sus características?* [En línea] https://ayudaleyprotecciondatos.es/bases-de-datos/relacional/.
13. **McKeown, Rebecca.** LearnSQL. *SQL INSERT, SQL UPDATE, SQL DELETE – Oh My!* [En línea] LearnSQL, 3 de 1 de 2020. https://learnsql.com/blog/sql-insert-sql-update-sql-delete-oh-my/.
14. **Amazon.** AWS. *¿Qué son las bases de datos NoSQL?* [En línea] Amazon. https://aws.amazon.com/es/nosql/.
15. **Wikipedia.** Wikipedia. *Arithmetic logic unit.* [En línea] 21 de 3 de 2022. https://en.wikipedia.org/wiki/Arithmetic_logic_unit.

16. **Simple 2 Code.** Simple 2 Code. *Block Diagram of a Computer.* [En línea] 30 de 9 de 2021. https://simple2code.com/computer-fundamentals/block-diagram-of-a-computer/.
17. **Júnior, Elemar.** Eximia Co. *Implementing parallel reduction in CUDA.* [En línea] 10 de 6 de 2019. https://eximia.co/implementing-parallel-reduction-in-cuda/.
18. **Black Hat Technology.** Segment Fault. *Understand the cache consistency protocol.* [En línea] 5 de 4 de 2020. https://segmentfault.com/a/1190000022267769.
19. **Priyadarshana, Ashan.** Medium. *CUDA — GPU Memory Architecture.* [En línea] 25 de 2 de 2018. https://ashanpriyadarshana.medium.com/cuda-gpu-memory-architecture-8c3ac644bd64.
20. **Pastor, Javier.** Xataka. *Antes teníamos la ley de Moore, ahora tenemos la 'ley de Huang' que perfila el futuro de NVIDIA y ARM.* [En línea] 21 de 9 de 2020. https://www.xataka.com/robotica-e-ia/antes-teniamos-ley-moore-ahora-tenemos-ley-huang-que-perfila-futuro-nvidia-arm.