UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

PLAN DE TRABAJO DE GRADO

FECHA DE PRESENTACIÓN: Bucaramanga, 9 de septiembre de 2025

TÍTULO: Estudio experimental sobre el impacto del uso de SYCL en el cálculo de Streamlines, Streaklines y Pathlines

MODALIDAD: Trabajo de investigación

AUTORES: Juan Pablo Cerinza Zaraza, 2190081
Ashley Michelle Calderon Villamizar, 2162101

DIRECTOR: Sergio Augusto Gélvez Cortes, Escuela De Ingeniería De Sistemas e Informática

ENTIDAD INTERESADA: Universidad Industrial de Santander

TABLA DE CONTENIDO

ESTUDIO EXPERIMENTAL SOBRE EL IMPACTO DEL USO DE SYCL EN EL CÁLCULO DE STREAMLINES, STREAKLINES Y PATHLINES

1 INTRODUCCIÓN

El desarrollo de la computación de alto rendimiento (High Performance Computing, HPC) ha permitido abordar problemas científicos y de ingeniería que requieren un procesamiento intensivo de datos. Dentro de este ámbito, la programación paralela constituye una estrategia esencial para aprovechar de manera eficiente las arquitecturas modernas de cómputo, particularmente aquellas basadas en unidades de procesamiento gráfico (GPU). Este enfoque resulta especialmente relevante en áreas como la dinámica de fluidos computacional (CFD, por sus siglas en inglés), en la cual el cálculo de trayectorias de partículas —representadas mediante Streamlines, Streaklines y Pathlines— demanda una elevada capacidad de cómputo y optimización en la gestión de recursos.

Entre las tecnologías predominantes en este campo, CUDA (Compute Unified Device Architecture) se ha consolidado como una herramienta de referencia debido a su estrecha integración con hardware de NVIDIA y a la madurez de su ecosistema de desarrollo. No obstante, la dependencia de un único proveedor limita su portabilidad y dificulta la adaptación a entornos heterogéneos. En respuesta a esta necesidad, ha emergido SYCL, un estándar desarrollado por el consorcio Khronos, que ofrece un modelo de programación en C++ para la ejecución paralela en dispositivos diversos (CPU, GPU, FPGA, entre otros). SYCL promueve la portabilidad y la interoperabilidad, características de creciente relevancia en la investigación y la industria, al tiempo que busca mantener un alto nivel de rendimiento.

El presente trabajo tiene como propósito realizar un estudio experimental sobre el impacto del uso de SYCL en el cálculo de Streamlines, Streaklines y Pathlines, comparando su rendimiento con una implementación equivalente desarrollada en CUDA. El análisis se centrará en métricas de tiempo de ejecución, utilización de memoria y eficiencia de cómputo, con el fin de establecer una comparación objetiva entre ambos enfoques. Para garantizar condiciones homogéneas, se emplearán herramientas de perfilamiento como NVIDIA Nsight Systems/Compute y entornos controlados mediante Docker, lo que permitirá asegurar la reproducibilidad de los experimentos.

Con ello, la investigación busca contribuir al conocimiento sobre la aplicabilidad de modelos de programación portables en escenarios de alto costo computacional, aportando evidencia que permita valorar la pertinencia de SYCL como alternativa a CUDA en el ámbito de la dinámica de fluidos y, en general, en el desarrollo de software científico de alto rendimiento.

2 PLANTEAMIENTO Y JUSTIFICACIÓN DEL PROBLEMA

La visualización de campos de flujo, mediante técnicas como Streamlines, Streaklines y Pathlines, es fundamental en diversas áreas de la ciencia e ingeniería, incluyendo la dinámica de fluidos computacional (CFD), la meteorología, y el modelado de procesos

físicos (Camp, Garth, Childs, Pugmire, y Joy, s.f.). Estos métodos permiten representar de forma intuitiva el comportamiento de partículas en movimiento dentro de un campo vectorial, lo cual facilita el análisis y la interpretación de fenómenos complejos.

Sin embargo, la simulación precisa y eficiente de estas trayectorias implica un alto costo computacional, especialmente cuando se trabaja con volúmenes de datos grandes o se requieren resoluciones espaciales y temporales finas (Pugmire, Childs, Garth, Ahern, y Weber, s.f.). En este contexto, la programación tradicional en C/C++, aunque eficiente, puede resultar insuficiente para explotar de manera óptima las capacidades de cómputo de los sistemas modernos, en especial aquellos que cuentan con aceleradores como GPUs.

En respuesta a esta necesidad, han surgido enfoques como la programación heterogénea, que busca distribuir la carga computacional entre distintos dispositivos de procesamiento, permitiendo una ejecución más rápida y eficiente. En particular, el estándar SYCL, basado en C/C++, ofrece una solución portable y flexible para desarrollar aplicaciones que aprovechen arquitecturas heterogéneas sin perder la expresividad y control del lenguaje de bajo nivel (Keryell, s.f.).

A pesar de sus beneficios teóricos, existe una brecha de conocimiento práctico respecto al impacto real del uso de SYCL en problemas computacionalmente intensivos como el cálculo de líneas de flujo. No se cuenta con suficiente evidencia comparativa que cuantifique su desempeño frente a enfoques tradicionales en términos de tiempo de ejecución, consumo de memoria y escalabilidad, con respecto a esta técnica de visualización en particular. Para cuantificar el desempeño se realizan estudios experimentales usando métricas como tiempo de ejecución, uso de memoria y escalabilidad (Camp, Krishnan, y cols., s.f.), relacionadas con características del problema como: tamaño del dataset, tamaño del conjunto de semillas, distribución del conjunto de semillas y complejidad del campo vectorial (Camp, Garth, y cols., s.f.).

3 OBJETIVOS

3.1 OBJETIVO GENERAL

- Comparar el desempeño computacional de una técnica de visualización de campos de flujo (Streamlines, Streaklines y Pathlines), que involucra el método de Runge-Kutta implementado en lenguaje C/C++, mediante métodos tradicionales de cómputo y programación heterogénea con SYCL, evaluando métricas como el tiempo de ejecución, uso de memoria y escalabilidad.

3.2 OBJETIVOS ESPECÍFICOS

Aunque los OE incluidos no hacen parte de esta evaluación, se hace la observación que los mismos están formulados como actividades y no como Objetivos. NOTA: Se ha de tener presente esta observación en la formulación del Plan

- Formular una versión optimizada del algoritmo RK4 para ejecución en arquitecturas heterogéneas utilizando SYCL.
- Comparar el comportamiento de las implementaciones bajo diferentes características del problema.
- Examinar los resultados obtenidos para determinar ventajas, limitaciones y condiciones óptimas de uso de SYCL frente a soluciones de solo CPU o de GPU de uso común en la industria como CUDA.

4 MARCO DE REFERENCIA

Como base para el desarrollo del proyecto es necesario establecer los fundamentos para la formulación del algoritmo optimzado de RK4, por lo cual es imperativo conocer las características del problema , los principios de la computación heterogénea y la computación de alto rendimiento, las aplicaciones de la misma en la industria y las partes requeridas para la optimización, las cuales se describen a continuación.

4.1

El concepto de computación autonómica, definido inicialmente por IBM (?), se refiere a un conjunto de características que presenta un sistema computacional el cual le permite actuar de manera autónoma, o auto-gobernarse, con el fin de alcanzar algún objetivo establecido por los administradores del sistema (?, ?).

Los 8 elementos clave, definidos por IBM, que deberían presentar este tipo de sistemas son:

1. Auto-conocimiento: habilidad de conocer su estado actual, las interacciones del sistema.
2. Auto-configuración: capacidad de reconfigurarse frente a los constantes cambios en el entorno.
3. Auto-optimización: búsqueda constante de optimizar el funcionamiento de sí mismo.
4. Auto-sanación: aptitud de restaurar el sistema en el caso de que se presenten fallas.
5. Auto-protección: facultad de protegerse a sí mismo de ataques externos.
6. Auto-conciencia: posibilidad de conocer el ambiente en el que el sistema se encuentra.
7. Heterogeneidad: capacidad de interactuar con otros sistemas de manera cooperativa.
8. Abstracción: ocultar la complejidad a los administradores del sistema con objetivos de alto nivel de abstracción.

En el caso de que un sistema tenga una implementación parcial de estas características, este podría considerarse autonómico. En este sentido debería tener la capacidad de lidiar con los problemas como la complejidad, heterogeneidad e incertidumbre (?, ?) al igual que reducir la cantidad de recursos tanto técnicos como humanos requeridos para mantener los sistemas en funcionamiento.

MAPE-K

IBM, en cuanto a la implementación de las características, propone una modelo de ciclo auto-adaptativo, denominado MAPE-K (?, ?). Este acercamiento, compuesto de cinco fases, es uno de ciclos de control más usado en implementaciones de sistemas auto-adaptativos y computación autonómica (?, ?). En la figura 1, se presentan las fases que *manejador* debe desarrollar para así administrar cada uno de los elementos del sistema computacional basado en una base de conocimiento común (?, ?).

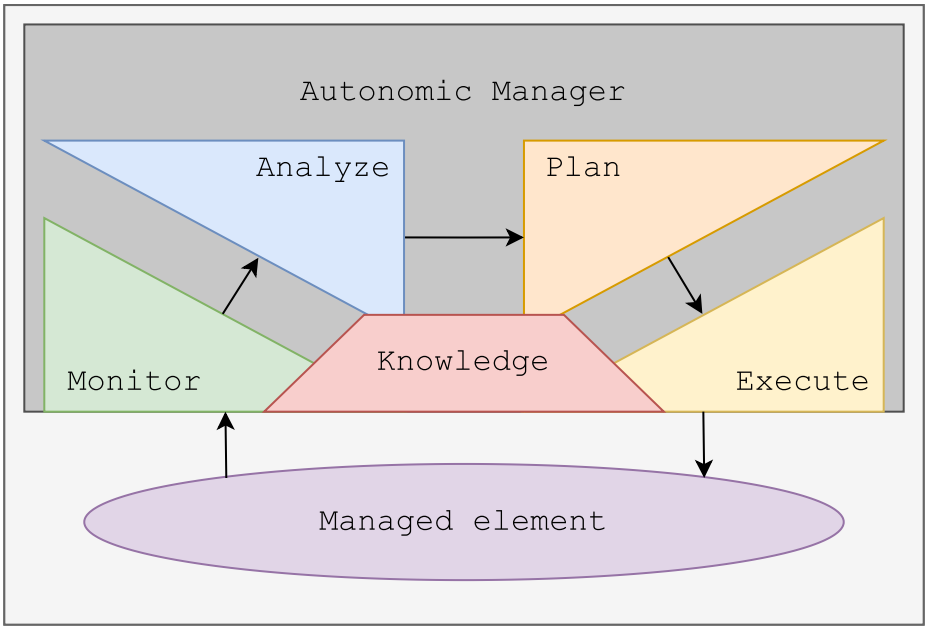

Figura 1: El ciclo auto-adaptativo MAPE-K.
(?, ?)

Cada una de estas fases son:

- Monitorear (M): Esta fase se compone de la recolección, filtración y reportar la información adquirida sobre el estado del elemento a manejar.
- Analizar (A): La fase de análisis se encarga del interpretar el entorno en el cual se encuentra, el predecir posibles situaciones comunes y diagnosticar el estado del sistema.

- Planear (P): Durante la planificación se determina las acciones a tomar con el fin de llegar a un objetivo establecido a partir de una serie de reglas o estrategias.
- Ejecutar (E): Finalmente, se ejecuta lo planeado usando los mecanismos disponibles para el manejo del sistema.

Es de resaltar que este modelo, aunque útil para el desarrollo de este tipo de sistemas, es bastante general en cuanto a la estructura y no usan modelos de diseño establecidos (?, ?).

MECANISMOS DE DESCRIPCIÓN

La fase de monitoreo dentro del ciclo MAPE-K es vital para el funcionamiento del manejador autonómico pues es a partir de la información que se construirá la base de conocimiento requerida por las demás partes del ciclo. Parte de esta, está compuesta por el *estado del sistema* el cual incluye la descripción del sistema en un momento dado (?, ?).

Existen varias maneras de realizar implementaciones de mecanismos de auto-descripción y la utilidad de cada uno de estos varía dependiendo en el tipo de sistema de software que se esté usando. Para el marco del proyecto, nos interesan aquellos que estén orientados a los sistemas embebidos e IoT, algunos de estos son:

- **JSON Messaging**: Iancu y Gatea (?) plantean un protocolo que emplea mensajería entre *gateways* con el fin de recibir información sobre estas. En términos simples, estas funcionan como un *ping* hacia el nodo que luego retorna sus datos, al igual que los dispositivos conectados a ella, al encargado de recolectar toda esta información con el fin de construir una descripción del sistema.-
- **IoT Service Description Model**: O IoT-LMsDM, es un servicio de descripción desarrollado por Zhen y Aiello (?) el cual está orientado al contexto, servicios e interfaz de un sistema IoT. De este se espera poder contar no solo con descripciones del estado del sistema en términos del ambiente, pero la funcionalidad (es decir, los *endpoints* a usar) al igual que las estructuras de datos que estos consumen.
- **Adaptadores de Auto-descripción**: En este acercamiento a los mecanismos de auto-descripción, se tienen adaptadores los cuales emplean los datos generados por los sensores del componente gestionado con el fin de realizar la determinación de la arquitectura desplegada. De igual manera, este acercamiento permite realizar modificaciones a la descripción de manera manual en caso de que se detecten problemas (?, ?).

De esto podemos ver no solo las diferentes maneras en las que las implementaciones realizan las descripciones de los sistemas asociados, sino que también el alcance de estos en cuanto a lo que pueden describir.

MECANISMOS DE ADAPTACIÓN

La adaptación, en el contexto de la computación autonómica, es la parte más importante en cuanto a la auto-gestión de un sistema de software se refiere. Así mismo, presenta el mayor reto debido a la necesidad de modificar código de bajo nivel, tener que afrontarse a incertidumbre de los efectos que pueden tener dichas alteraciones al sistema al igual que lidiar con esto en *runtime* debido a los problemas que el *downtime* tendría en los negocios (?, ?).

Esta adaptabilidad puede exponerse en múltiples puntos dentro de un sistema de software. Pueden realizarse adaptaciones en sistema operativo, lenguaje de programación, arquitectura e incluso datos (?, ?).

Manteniéndose en el marco del proyecto, son las implementaciones relacionadas con la modificación de la arquitectura los cuales nos interesan. Siendo así, nos centraremos en los mecanismos de adaptación de componentes, o de reconfiguración:

- **Binding Modification**: Este mecanismo hace referencia a la alteración de los vínculos entre los diferentes componentes de la arquitectura. Estos tienen el objetivo de modificar la interacción entre componentes, lo que es especialmente común en implementaciones con *proxies*. Este tipo de mecanismo de adaptación fue usado por Kabashkin (?) para añadir fiabilidad a la red de comunicación aérea.
- **Interface Modification**: Las interfases funcionan como los puntos de comunicación entre los diferentes componentes de la arquitectura. Siendo así, es posible que la modificación de estos sea de interés con el fin de alterar el comportamiento de un sistema al igual que soportar la heterogeneidad del sistema. Esto puede verse en el trabajo desarrollado por Liu, Parashar y Hariri (?) en donde definen la utilidad de dichas adaptaciones al igual que la implementación de las mismas.
- **Component Replacement, Addition and Subtraction**: En términos simples, este mecanismo se encarga de alterar los componentes que componen la arquitectura; de esta manera, modificando su comportamiento. Ejemplos de esto puede verse en el trabajo de Huynh (?) en el cual se evalúan varios acercamientos a la reconfiguración de arquitecturas a partir del remplazo de componentes a nivel individual al igual que grupal.

  Este acercamiento a la mutación de la arquitectura también puede verse en el despliegue de componentes como respuesta a cambios en los objetivos de negocio de las aplicaciones al igual que como respuesta a cambios inesperados dentro de la aplicación. Esto puede verse en trabajos como el de Patouni (?) donde se realizan este tipo de implementaciones.

4.2 INTERNET OF THINGS

El Internet de las cosas, o IoT (Internet of Things); es una de las áreas de las ciencias de la computación en la cual se embeben diferentes dispositivos en objetos del día a día. Esto les

da la capacidad de enviar y recibir información con el fin de realizar monitoreo o facilitar el control de ciertas acciones (?, ?).

Esta tecnología, debido a su flexibilidad al igual que el alcance que puede tener, presenta una gran cantidad de aplicaciones que va desde electrónica de consumo hasta la industria. Encuestas realizadas en el 2020 reportan su uso en smart homes, smart cities, transporte, agricultura, medicina, etc. (?, ?). Su impacto no ha sido poco.

SISTEMAS EMBEBIDOS

Los sistemas de cómputo embebidos hacen referencia a un sistema compuesto de microcontroladores los cuales están orientados a llevar a cabo una función o un rango de funciones específicas (?, ?). Este tipo de sistemas, debido a la posibilidad de combinar hardware y software en una manera compacta, se ha visto en multiples campos de la industria como lo son el sector automotor, de maquinaria industrial o electrónica de consumo (?, ?).

SMART CAMPUS

Un Smart Campus, equiparable con el concepto de Smart City, es una plataforma en la que se emplean tecnologías, sumado a una infraestructura física, con la cual se busca la recolección de información y monitoreo en tiempo real (?, ?). Los datos recolectados tienen el objetivo de apoyar la toma de decisiones, mejora de servicios, entre otros (?, ?).

Estas plataformas, debido a su escala y alcance en cuanto a la cantidad de servicios que pueden ofrecer, requieren de infraestructuras tecnológicas las cuales den soporte a los objetivos del sistema. Es posible ver implementaciones orientadas a microservicios en trabajos como los de Jiménez, Cárcamo y Pedraza (?) donde se desarrolla una plataforma de software escalable con la cual se pueda lograr interoperatividad y alta usabilidad para todos.

4.3 DOMAIN-SPECIFIC LANGUAGES

Los *Domain-Specific Languages* (DSL), o Lenguaje específico de dominio, son lenguajes de programación los cuales se usan con un fin específico. Estos están orientados principalmente al uso de abstracciones de un mayor nivel debido al enfoque que estos tienen para la solución de un problema en específico (?, ?). Ejemplos de este tipo de acercamiento, en el caso del modelado (*Domain-specific modeling*), pueden observarse en los diagramas de entidad relación usados en el desarrollo de modelos de base de datos (?, ?).

5 METODOLOGÍA

Para el desarrollo del trabajo de grado, se propone un modelo de prototipado iterativo compuesto de 5 fases (Ver fig. 2). De esta manera, se avanzará a medida que se va completando la fase anterior y permitirá a futuro el poder iterar sobre lo que se ha desarrollado anteriormente.

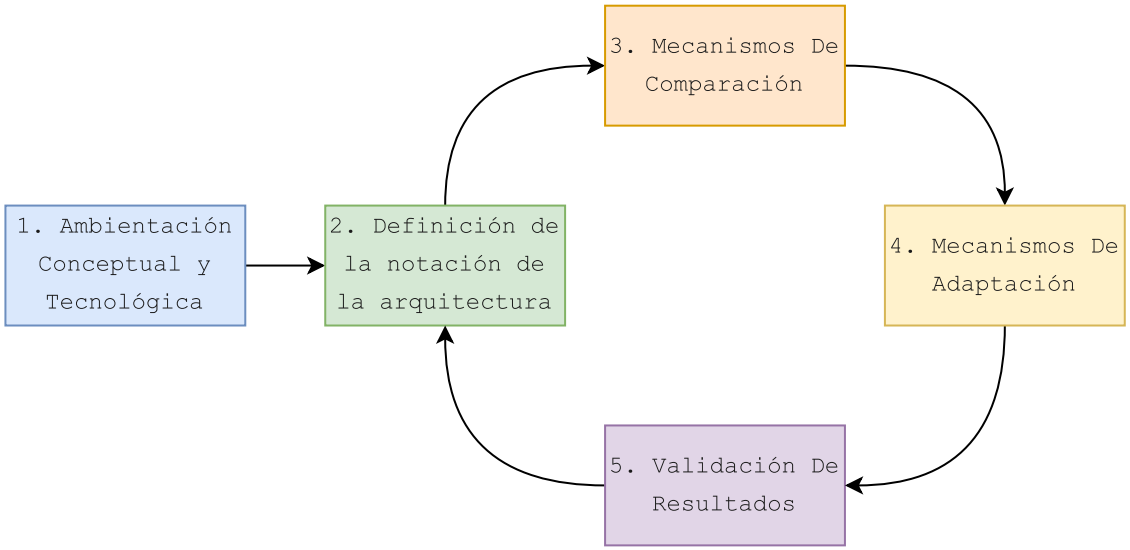

Figura 2: Metodología del proyecto

5.1 AMBIENTACIÓN CONCEPTUAL Y TECNOLÓGICA

La primera fase de la metodología se basa en la investigación de la literatura, al igual que de la industria, necesaria para cubrir las bases tanto conceptuales como técnicas necesarias para el desarrollo del proyecto.

ACTIVIDADES

5.1.1. Identificación de las características principales de un sistema auto-adaptable.

5.1.2. Análisis de los mecanismos de adaptación de la arquitectura.

5.1.3. Análisis los algoritmos empleados para la comparación de la comparación de las arquitecturas.

5.1.4. Determinación de los criterios de selección para el lenguaje de notación.

5.1.5. Evaluación de los posibles lenguajes de programación para la implementación a realizar.

5.1.6. Análisis, retroalimentación y conclusiones del desarrollo de la fase.

5.2 DEFINICIÓN DE LA NOTACIÓN DE LA ARQUITECTURA

La segunda fase está en la definición del cómo se realiza la declaración de la arquitectura. Partiendo de los criterios de selección establecidos en la fase 1, se espera determinar un lenguaje de notación el cual nos permita definir la arquitectura objetivo a alcanzar, al igual que la gramática correspondiente para poder realizar dicha declaración.

ACTIVIDADES

5.2.1. Selección del lenguaje de marcado a usar a partir de los criterios establecidos.

5.2.2. Definición la gramática a usar para la definición de la arquitectura.

5.2.3. Implementación la traducción de la notación al modelo de grafos.

5.2.4. Determinación como se realizará la representación de los componentes y partes de la arquitectura.

5.2.5. Análisis, retroalimentación y conclusiones del desarrollo de la fase.

5.3 MECANISMOS DE COMPARACIÓN

Durante la tercera fase del proyecto, se buscará poder determinar e implementar cómo se realizará la comparación entre el estado de la arquitectura obtenido durante la auto-descripción de la misma y el objetivo establecido. Así mismo, y con el fin de reportar a los administradores de los sistemas, también será necesario definir *niveles* de similitud entre las 2 arquitecturas.

ACTIVIDADES

5.3.1. Selección del mecanismo de comparación a usar para evaluación de estado de la arquitectura.

5.3.2. Implementación del mecanismo de comparación seleccionado.

5.3.3. Determinación de los diferentes niveles de similitud entre arquitecturas.

5.3.4. Análisis, retroalimentación y conclusiones del desarrollo de la fase.

5.4 MECANISMOS DE ADAPTACIÓN

La cuarta fase del proyecto está orientada a la selección, al igual que la implementación en Smart Campus UIS, del conjunto de mecanismos de adaptación de la arquitectura.

ACTIVIDADES

5.4.1. Definición el conjunto de mecanismos de adaptación.

5.4.2. Implementación el conjunto de mecanismos de adaptación seleccionados.

5.4.3. Análisis, retroalimentación y conclusiones del desarrollo de la fase.

5.5 VALIDACIÓN DE RESULTADOS

La fase final del proyecto se encargará principalmente de la realización de pruebas de los mecanismos implementados, los resultados obtenidos al igual que la documentación de todo lo que se desarrolló durante el proyecto.

ACTIVIDADES

5.5.1. Realización de las pruebas del funcionamiento de la implementación realizada con diversas arquitecturas objetivo.

5.5.2. Recopilación la documentación generada durante el desarrollo de cada una de las fases del proyecto.

5.5.3. Compilación de la documentación para generar el documento de final.

5.5.4. Correcciones y adiciones para la presentación final del proyecto de grado.

6 CRONOGRAMA

En la tabla 1 se presenta el cronograma propuesto para el desarrollo del el proyecto. En este se establece un tiempo total de 16 semanas en las cuales se desarrollarán las actividades definidas en la metodología.

Fase / Semana	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Ambientación Conceptual Y Tecnológica																
Definición De La Notación De La Arquitectura																
Mecanismos De Comparación																
Mecanismos De Adaptación																
Validación De Resultados																
Fase: Ambientación Conceptual Y Tecnológica																
Activadad / Semana	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
5.1.1.																
5.1.2.																
5.1.3.																
5.1.4.																
5.1.5.																
5.1.6.																
Fase: Definición De La Notación De La Arquitectura																
Activadad / Semana	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
5.2.1.																
5.2.2.																
5.2.3.																
5.2.4.																
5.2.5.																
Fase: Mecanismos De Comparación																
Activadad / Semana	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
5.3.1.																
5.3.2.																
5.3.3.																
5.3.4.																
Fase: Mecanismos De Adaptación																
Activadad / Semana	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
5.4.1.																
5.4.2.																
5.4.3.																
Fase: Validación De Resultados																
Activadad / Semana	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
5.5.1.																
5.5.2.																
5.5.3.																
5.5.4.																

Tabla 1: Cronograma del proyecto

7 PRESUPUESTO

GASTOS DE PERSONAL			
INVESTIGADOR	**FUNCIÓN**	**DEDICACIÓN**	**TOTAL**
PHD. GABRIEL RODRIGO PEDRAZA FERREIRA*	DIRECTOR	16 HRS	$4,880,000.00
MSC. HENRY ANDRÉS JIMÉNEZ HERRERA*	CO-DIRECTOR	16 HRS	$4,080,000.00
EST. DANIEL DAVID DELGADO CERVANTES**	AUTOR	728∼1000 HRS	$1,160,000.00
TOTAL			$10,120,000.00

RUBRO	**VALOR**
GASTOS DEL PERSONAL	$10,120,000.00
GASTOS DE EQUIPOS**	$6,317,405.00
GASTOS DE MATERIAL*	$200,000.00
GASTOS DE PUBLICACION**	$400,000.00
TOTAL	$17,037,405.00

8 BIBLIOGRAFÍA

Camp, D., Garth, C., Childs, H., Pugmire, D., y Joy, K. (s.f.). Streamline integration using MPI-hybrid parallelism on a large multicore architecture. , *17*(11), 1702–1713. Descargado 2025-09-10, de http://ieeexplore.ieee.org/document/5669297/ doi: 10.1109/TVCG.2010.259

Camp, D., Krishnan, H., Pugmire, D., Garth, C., Johnson, I., Bethel, E. W., ... Childs, H. (s.f.). *GPU acceleration of particle advection workloads in a parallel, distributed memory setting.* The Eurographics Association. Descargado 2025-09-10, de http://diglib.eg.org/handle/10.2312/EGPGV.EGPGV13.001-008 (Artwork Size: 8 pages ISBN: 9783905674453 ISSN: 1727-348X Pages: 8 pages Publication Title: Eurographics Symposium on Parallel Graphics and Visualization) doi: 10.2312/EGPGV/EGPGV13/001-008

Keryell, R. (s.f.). *SYCL a single-source c++ standard for heterogeneous computing.* Descargado de https://www.khronos.org/assets/uploads/developers/presentations/SC19-H2RC-keynote-SYCL_Nov19.pdf

Pugmire, D., Childs, H., Garth, C., Ahern, S., y Weber, G. H. (s.f.). Scalable computation of streamlines on very large datasets. En *Proceedings of the conference on high performance computing networking, storage and analysis* (pp. 1–12). ACM. Descargado 2025-09-10, de https://dl.acm.org/doi/10.1145/1654059.1654076 doi: 10.1145/1654059.1654076