Universidad Nacional Federico Villarreal

VRIN | VICERRECTORADO DE INVESTIGACIÓN

ESCUELA UNIVERSITARIA DE POSGRADO

MOTOR DE INFERENCIA PARALELO EN UNA ARQUITECTURA HETEROGÉNEA (MULTINÚCLEO Y GPGPU) PARA LA OPTIMIZACIÓN EN LA DEDUCCIÓN DE INFORMACIÓN IMPLÍCITA EN LAS BASES DE DATOS

Línea de investigación:

Ingeniería de software, simulación y desarrollo de TICs

Tesis para optar el grado académico de Doctor en Ingeniería de Sistemas

Autor:

Rozas Huacho, Javier Arturo

Asesor:

Quispe Prado, Wilber

(ORCID: 0000-0003-2452-3669)

Jurado:

Rodriguez Rodriguez, Ciro

Esenarro Vargas, Doris

Sobrado Gómez, Angel

Lima - Perú

2024

MOTOR DE INFERENCIA PARALELO EN UNA ARQUITECTURA HETEROGÉNEA (MULTINÚCLEO Y GPGPU) PARA LA OPTIMIZACIÓN EN LA DEDUCCIÓN DE INFORMACIÓN IMPLÍCITA EN LAS BASES DE DATOS

INFORME DE ORIGINALIDAD

9%	9%	2%	0%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	repositorio.unfv.edu.pe Fuente de Internet	1%
2	hdl.handle.net Fuente de Internet	1%
3	albertovillalobos1.wordpress.com Fuente de Internet	1%
4	dokumen.pub Fuente de Internet	1%
5	www.slideshare.net Fuente de Internet	<1%
6	www.coursehero.com Fuente de Internet	<1%
7	www.gob.pe Fuente de Internet	<1%
8	docplayer.es Fuente de Internet	<1%

Universidad Nacional
Federico Villarreal

VRIN | VICERRECTORADO DE INVESTIGACIÓN

ESCUELA UNIVERSITARIA DE POSGRADO

MOTOR DE INFERENCIA PARALELO EN UNA ARQUITECTURA HETEROGÉNEA (MULTINÚCLEO Y GPGPU) PARA LA OPTIMIZACIÓN EN LA DEDUCCIÓN DE INFORMACIÓN IMPLÍCITA EN LAS BASES DE DATOS

Línea de Investigación:
Ingeniería de software, simulación y desarrollo de TICs

Tesis para optar el grado académico de Doctor en Ingeniería de Sistemas

Autor:
Rozas Huacho, Javier Arturo

Asesor:
Quispe Prado, Wilber
(ORCID: 0000-0003-2452-3669)

Jurado:

Rodriguez Rodriguez, Ciro
Esenarro Vargas, Doris
Sobrado Gómez, Angel

Lima – Perú

2024

Dedicatoria

A Zonia por compartir sueños y toda una vida.

A Taly, Nay y Javi presencia de Dios en mi vida.

A mis padres (+) energía presente en el día a día de mi vida.

A mis hermanos, familia y amigos, que me permiten transitar este mundo con alegría y optimismo.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El objetivo fue optimizar la deducción de información implícita en las bases de datos, mediante el uso de un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU). Se ideó, diseñó e implementó una propuesta de un modelo de un motor de inferencia paralelo en una arquitectura heterogénea; en base a sentencias lógicas expresadas en forma normal conjuntiva (FNC), soportadas en estructuras de datos unidimensionales. La validación se realizó mediante la implementación de un prototipo, con el que se experimentó ejecutando procesos de inferencia y disminuyó tiempos de ejecución. Se logró una optimización de hasta en un 72.42% en una arquitectura multinúcleo y de hasta 93.01 en una arquitectura GPGPU; lo que, permitió incrementos de eficiencia (medida con la métrica SpeedUp), superior a 3 veces en una arquitectura multinúcleo y superior a 14 veces en una arquitectura GPGPU. Para la proyección de tiempos de procesamiento se logró correlacionar a funciones polinómicas de grado 3, con un coeficiente de determinación de 0.99. En conclusión, los porcentajes de optimización e incrementos de eficiencia obtenidos con la propuesta del motor de inferencia paralelo, son bastante aceptables, que viabilizan su inclusión como parte de los sistemas de gestión administrativa.

Palabras clave: motor de inferencia, bases de datos deductivas, algoritmos paralelos, arquitectura heterogénea, procesadores multinúcleo, GPGPU, iGPU, dGPU.

ABSTRACT

The objective was to optimize the deduction of implicit information in databases, through the use of a parallel inference engine in a heterogeneous architecture (multicore and GPGPU). A proposed model of a parallel inference engine on a heterogeneous architecture was devised, designed and implemented; based on logical sentences expressed in conjunctive normal form (CNF), supported by one-dimensional data structures. The validation was carried out through the implementation of a prototype, which was experimented with by executing inference processes and reduced execution times. An optimization of up to 72.42% was achieved on a multicore architecture and up to 93.01 on a GPGPU architecture; which allowed increases in efficiency (measured with the SpeedUp metric) of more than 3 times in a multicore architecture and more than 14 times in a GPGPU architecture. For the projection of processing times, it was possible to correlate it to polynomial functions of degree 3, with a determination coefficient of 0.99. In conclusion, the optimization percentages and efficiency increases obtained with the proposal of the parallel inference engine are quite acceptable, which makes its inclusion feasible as part of the administrative management systems.

Key words: inference engine, deductive databases, parallel algorithms, heterogeneous architecture, multi-core processors, GPGPU, iGPU, dGPU.

I. INTRODUCCIÓN

En el trabajo se consigna el diseño de un motor de inferencias con algorítmica paralela que permita explicitar datos implícitos existentes en las bases de datos transaccionales de las organizaciones; dándole un valor agregado a estas bases de datos, permitiendo extender funcionalidad al nivel de deducción. Todo esto soportado en una arquitectura de procesamiento heterogénea, realizando análisis de rendimiento en arquitectura mono procesador, multinúcleo y en unidades de procesamiento gráfico de propósito general (GPGPU); considerando ambos tipos: integradas (iGPU) y dedicadas (dGPU). La perspectiva es coadyuvar a un nuevo paradigma en las aplicaciones de gestión administrativa, de modo que se propenda al uso de toda la capacidad de cómputo existente en las computadoras actuales; permitiendo así añadir al software de gestión convencional existente, capacidad de inferencia en sus diferentes funcionalidades, de modo que coadyuve a automatizar aún más los procesos, minimizando la intervención humana en puntos donde la aplicación pueda tomar decisiones, basadas en mecanismos determinísticos de inferencia.

El documento ha sido organizado acorde a la normatividad vigente planteada por la Universidad. En la primera sección, se efectúa un análisis del contexto actual y se identifica el problema, detallando los diferentes tópicos que corresponden a la Introducción de un documento académico, tal como: Descripción y formulación del problema, los antecedentes, la justificación, las limitaciones, los objetivos y la hipótesis.

En la siguiente sección, se considera los conceptos teóricos que dan soporte al presente trabajo, conceptos tales como: Representación del conocimiento, motores de inferencia, algoritmos paralelos, complejidad de algoritmos paralelos, computación heterogénea y sistemas de bases de datos deductivas.

En la siguiente sección, se describe la metodología utilizada en el trabajo, considerando tópicos como: Tipo de investigación, población y muestra, operacionalización de variables y el proceso de análisis de datos.

En las siguientes dos secciones se consideran, la realización del proceso experimental, el compendio de los resultados obtenidos, la contrastación de la hipótesis y la discusión de los resultados.

Finalmente, se consigna las conclusiones, recomendaciones y las referencias bibliográficas; concluyéndose con los anexos, donde principalmente se considera la matriz de consistencia, así como los algoritmos implementados en OpenCL que da soporte al procesamiento paralelo de los diferentes dispositivos de la arquitectura heterogénea.

1.1 Planteamiento del Problema

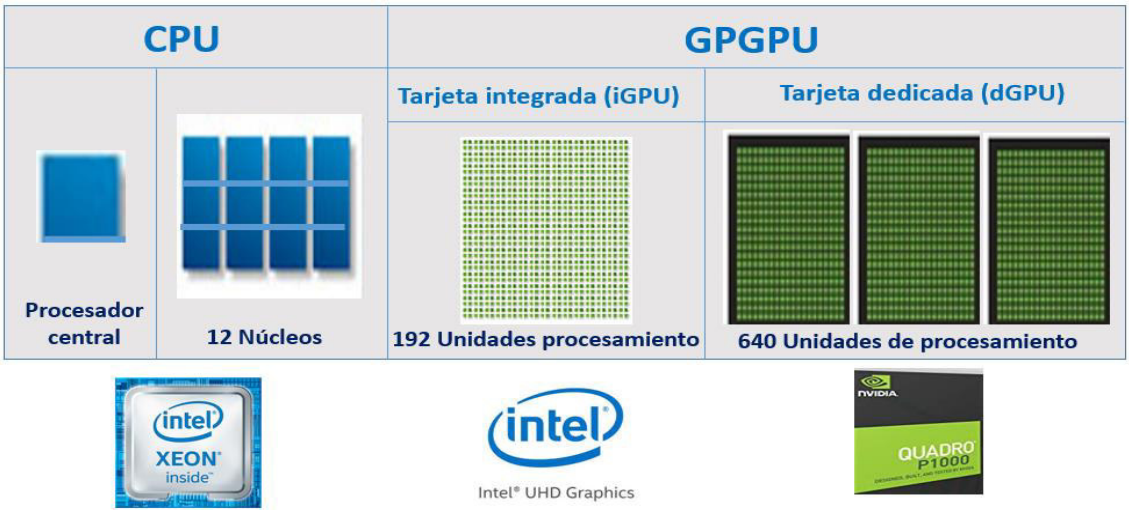
Asistimos a un mundo donde los avances tecnológicos no dejan de sorprendernos. El impacto de las tecnologías de la información y la comunicación (TIC), se percibe en las transformaciones trascendentales en la sociedad. Autores como: Friedman (2006) en su libro La Tierra es Plana, Zanoni (2014) en su libro Futuro Inteligente, Harari (2018) en su libro 21 Lecciones para el siglo XXI, Etc. Ponen de manifiesto la restructuración que sufrió, sufre y sufrirá la sociedad, como consecuencia de los incesantes y vertiginosos avances de las TIC's. Cada vez emergen nuevos conceptos asociados a estas tecnologías; se habla de inteligencia de negocios (Business Intelligence), inteligencia analítica (Analytic Intelligence), realidad virtual (virtual reality), realidad aumentada (augmented reality), cadena de bloques (blockchain), ciencia de datos, desarrollos impresionantes en inteligencia artificial como el aprendizaje automático (Machine learning y Deep learning) y la interconexión de los objetos denominada internet de las cosas (IoT – Internet of Things); permiten avizorar un futuro de una revolución tecnológica sin precedentes, donde los procesos de automatización serán el común denominador de nuestras actividades diarias.

Paralelamente, como se puede apreciar en Thomasian (2021), también se vienen dando avances vertiginosos en la tecnología hardware, que han venido evolucionando, desde computadoras con procesadores de diferentes tipos, como los simples “central processing unit” (CPU), hasta computadoras con “graphics processing unit” (GPU), “field-programmable gate array” (FPGA) o “Aplication Specific Integrate Circuit” (ASIC). Los procesadores del tipo CPU y GPU se encuentran en cualquier computador actual, mientras que los del tipo FPGA y ASIC actualmente se utilizan para propósitos muy específicos y no son de uso general, menos aún en contextos empresariales.

Estos avances vertiginosos de la tecnología tanto a nivel de software y hardware, no son explotados en los sistemas de gestión administrativa de las organizaciones, donde normalmente se toman decisiones con información mínima, desperdiciándose la información implícita que subyace en las bases de datos transaccionales, data warehouses o data lakes, debido al desconocimiento de tecnologías que permitan explicitar esta información. Por otra parte, la innovación en hardware de arquitectura de procesamiento heterogénea con la que cuentan todas las computadoras, que además del procesador central ofrecen la arquitectura multinúcleo y unidades de procesamiento gráfico de propósito general (GPGPU), que generalmente consideran dos tipos: integradas (iGPU) y dedicadas (dGPU); hace que esta alta capacidad de cómputo sea generalmente es subutilizada o desperdiciada. La figura 1 muestra la capacidad de cómputo de una Laptop Workstation HP ZBook 15 G5 (Intel(R) E-2176M CPU @ 2.70Ghz), donde se tiene 1 procesador central 12 núcleos lógicos y 832 unidades de procesamiento GPGPU, total 845 unidades de cómputo.

Figura 1

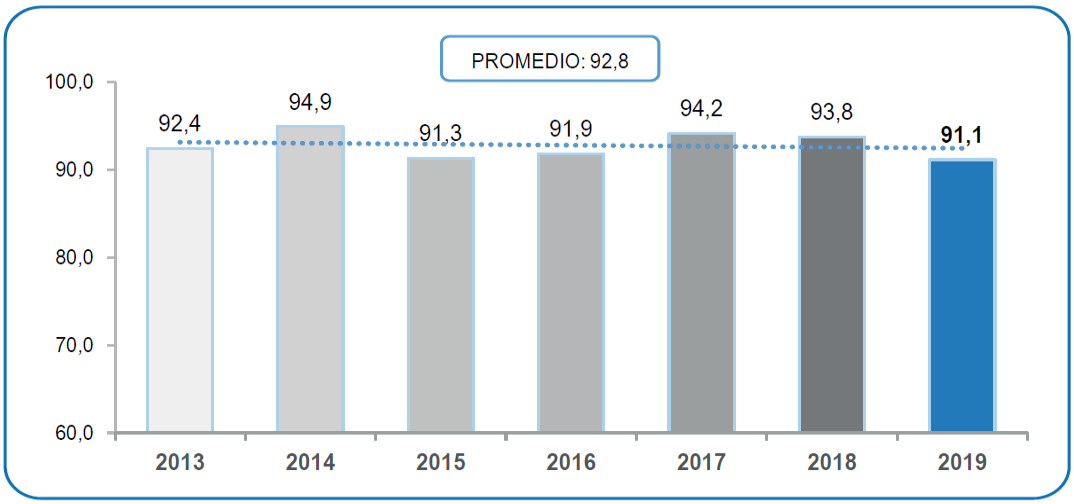
Capacidad de Cómputo de una Laptop Workstation HP ZBook 15 G5



La pregunta crucial en este punto es, ¿En qué medida estas tecnologías han tenido una penetración social en el contexto tecnológico peruano?, ¿Existe aplicaciones software con estas tecnologías en las organizaciones de nuestro país?, ¿Qué actitudes deben emprender las organizaciones responsables del desarrollo tecnológico del país?, ¿Es necesario que el software de gestión administrativa incluya componentes de inteligencia artificial?, ¿Qué rol debe asumir la educación superior en la difusión y uso de estas tecnologías?, etc. En el Perú, se evidencia que las organizaciones asumen que las tecnologías de la información juegan un papel preponderante en su desarrollo, de ahí que invierten en tecnología, tal como se indica en el documento “Perú: Tecnologías de Información y Comunicación en las Empresas, 2019” del Instituto Nacional de Estadística e Informática (2018), donde se ve que en gran porcentaje las empresas han utilizado equipos de cómputo durante cinco años consecutivos; con incremento en algunos años, tal como se percibe en la figura 2:

Figura 2

Empresas Peruanas que utilizaron computadoras entre el 2013 - 2019

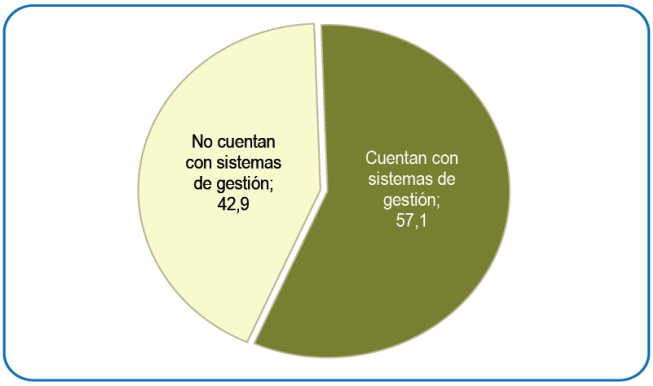


Nota. Recuperado del documento "Perú: Tecnologías de Información y Comunicación en las Empresas, 2019" del INEI (p. 83).

La figura 3 ilustra el porcentaje de las empresas del Perú que usan sistemas de gestión.

Figura 3

Empresas Peruanas según uso de sistemas de gestión entre el 2013 - 2019

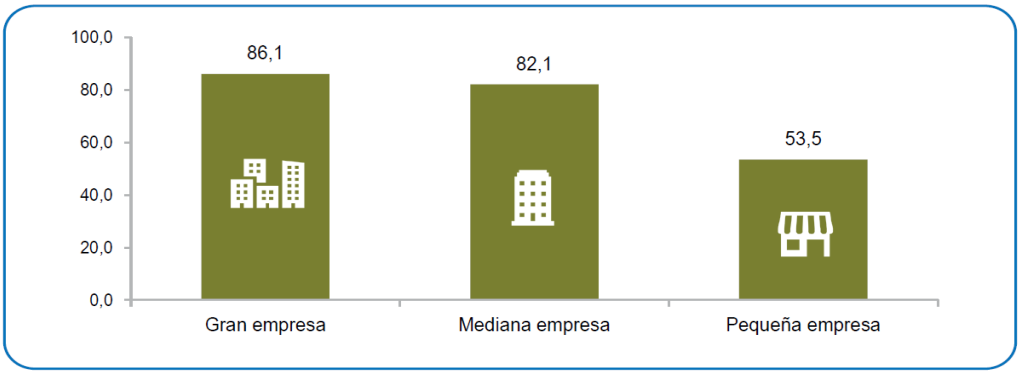


Nota. Recuperado del documento "Perú: Tecnologías de Información y Comunicación en las Empresas, 2019" del INEI (p. 54).

Respecto al uso de sistemas de gestión en los diferentes segmentos económicos empresariales, se puede apreciar los porcentajes correspondientes en la figura 4:

Figura 4

Empresas Peruanas según tipo de sistemas de gestión utilizadas entre el 2013 - 2019

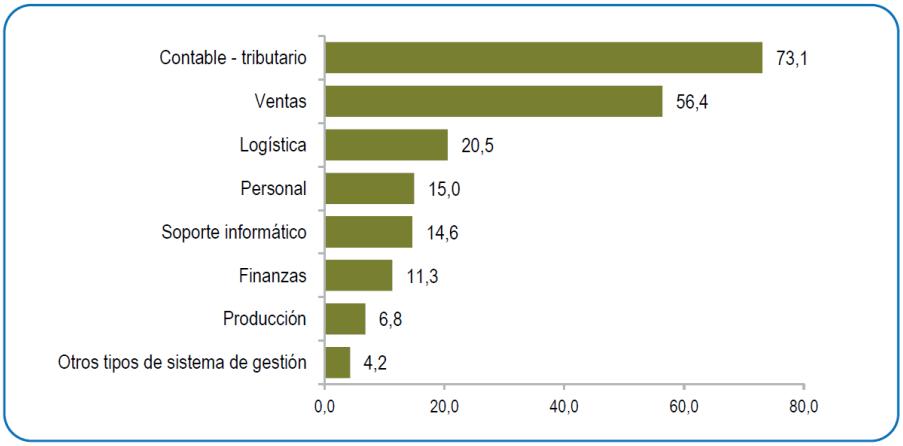


Nota. Recuperado del documento "Perú: Tecnologías de Información y Comunicación en las Empresas, 2019" del INEI (p. 55).

De los dos gráficos anteriores se infiere que las empresas utilizan mayoritariamente software de gestión; sin embargo, aún falta analizar el nivel de software de gestión que se utiliza, tal como se puede apreciar en la figura 5:

Figura 5

Empresas Peruanas según tipo de sistemas de gestión utilizadas entre el 2013 – 2019



Nota. Recuperado del documento "Perú: Tecnologías de Información y Comunicación en las Empresas, 2019" del INEI (p. 55).

En consecuencia, en nuestra realidad si existe niveles de automatización y uso de las tecnologías de la información; pero, la presencia de estas al interior de las organizaciones no está acorde con los nuevos paradigmas emergentes en tecnología.

El proceso de adopción e integración de estas tecnologías emergentes en los sistemas nerviosos digitales de las organizaciones se debe dar desde diferentes perspectivas, por ejemplo: Inclusión de componentes de inteligencia artificial, integración de internet de las cosas y explotación de grandes volúmenes de datos. De hecho, que este proceso se debe dar de manera gradual; inicialmente se puede considerar el procesamiento de los grandes volúmenes de datos, donde subyace información implícita que puede ser evidenciada y utilizada en línea y obtener así, ventajas competitivas; aprovechando prioritariamente, la gran capacidad de cómputo de la infraestructura hardware con la que cuentan actualmente las organizaciones, que es utilizada mínimamente; circunscribiéndose sólo al uso del procesador principal, desperdiciando el potencial de cómputo de los multinúcleos y los GPGPU.

Es en este contexto, que se enmarca el presente proyecto, en la intención de promover la computación paralela que permita maximizar el potencial de cómputo que tienen los actuales computadores; para que, en base a motores de inferencia paralelos, coadyuve a la explotación de la información implícita que subyace en las cada vez más voluminosas bases de datos transaccionales; orientado a la deducción determinística de información, que permita minimizar la intervención humana en los procesos de automatización de algunas actividades.

1.2 Descripción del Problema

Considerando la información de la Encuesta Económica Anual del INEI (2018), referida al uso de tecnología, se percibe que las organizaciones prioritariamente cuentan con sólo un primer nivel de informatización; donde, preponderantemente utilizan software de gestión que automatiza la parte operativa de la organización; como, por ejemplo, los sistemas de: contabilidad, logística, facturación, planillas, cuentas bancarias, etc. Estos subsistemas se

circunscriben a efectuar sólo procesos que implican: registro de información, cálculos y emisión de reportes.

Así mismo, las bases de datos que subsume el uso de motores de inferencia determinísticos que permiten deducir información implícita, no son conceptos nuevos; trabajos asociados a estos conceptos se han venido dando desde décadas anteriores, para lo cual existe abundante literatura, con propuestas cada vez más interesantes. Sin embargo, la incidencia de éstos en las aplicaciones software de gestión actual, soportados en el hardware multinúcleo y GPGPU existente en las organizaciones, es mínima o nula. Aunque se perfila como una nueva tendencia, que se ve fortalecida en la medida en que el hardware ofrece capacidad heterogénea de cómputo, cada vez más potente y barato.

Enfatizando, a pesar de la avasallante innovación tecnológica proporcionada por las TIC's, tanto a nivel algorítmico (inteligencia artificial) como a nivel de hardware (computación heterogénea); aún, las organizaciones no hacen uso o subutilizan estas innovaciones. No se percibe la importancia de incrementar funcionalidad a sus sistemas informáticos; por ejemplo, incorporando técnicas estocásticas propias de la minería de datos; o técnicas determinísticas como motores de inferencia que permitan aprovechar la información implícita que subyace en las bases de datos transaccionales.

Un aspecto relevante en algunos procesos de optimización es automatizar completamente algunas actividades, orientado en lo posible a prescindir de intervención humana. Es en estos contextos que se requiere de procesos determinísticos de manera imperativa. No hacerlo, implica generalmente que se incurren en mayores costos de tiempo y esfuerzo.

En suma, los sistemas de software de las empresas no consideran la explotación de sus datos transaccionales mediante motores de inferencia determinísticos, que permitan incrementar la funcionalidad de estos sistemas de información, aplicando las tecnologías

emergentes a nivel de hardware; como son las computadoras de arquitectura heterogénea, que incorporan múltiples núcleos, tarjetas integradas iGPU y tarjetas dedicadas dGPU.

1.3 Formulación del Problema

- **Problema general**

¿En qué medida se logra la optimización en la deducción de información implícita en las bases de datos, mediante el uso de un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU)?

- **Problemas específicos**

1. ¿Qué características debe tener un motor de inferencia paralelo en una arquitectura heterogénea, que permita optimizar la deducción de información implícita en las bases de datos?
2. ¿En qué medida disminuye el tiempo de la optimización en la deducción de información implícita en las bases de datos, mediante el uso de un motor de inferencia paralelo en una arquitectura heterogénea?
3. ¿En qué medida incrementa la eficiencia de la optimización en la deducción de información implícita en las bases de datos, mediante el uso de un motor de inferencia paralelo en una arquitectura heterogénea?
4. ¿En qué medida se puede proyectar los tiempos de procesamiento para cualquier volumen de datos, en la deducción de información implícita en las bases de datos, mediante el uso de un motor de inferencia paralelo en una arquitectura heterogénea?

1.4 Antecedentes de la Investigación

Existen antecedentes del uso de arquitecturas heterogéneas en la solución de problemas de diferentes áreas de la informática, tanto en bases de datos como en inteligencia artificial. La mayoría de estos antecedentes están orientados a la solución estocástica de problemas, que sirven de base para la toma de decisiones. Sin embargo, en la práctica en muchos problemas se

requiere de soluciones determinísticas; sobre todo aquellas que automatizan procesos y que minimizan o neutralizan en lo posible la intervención humana.

Considerando que el presente proyecto se enfoca a la solución determinística de explicitar información implícita mediante mecanismos de inferencia, en un contexto de bases de datos transaccionales; en esta sección, se consideran antecedentes que en alguna medida consideran aspectos de arquitecturas heterogéneas con bases de datos.

Gowanlock et al. (2021), en un trabajo de investigación publican un estudio del trabajo en la distribución y el almacenamiento de las bases de datos, en cuanto se refiere a las primitivas en arquitecturas heterogéneas del tipo CPU/GPU. Parten de la premisa que las tarjetas de procesamiento gráfico (GPU) además de tener núcleos independientes de procesamiento, vienen con grandes cantidades de memoria adicionales en la tarjeta; así como con anchos de banda amplios que permiten hacer frente al trabajo de la carga intensiva de datos. Asumen también, que, para optimizar la eficiencia de los algoritmos, se debe utilizar simultáneamente arquitectura heterogénea CPU/GPU; por ejemplo, al realizar las consultas.

Para el propósito del estudio, se seleccionó algoritmos de proceso intensivo de datos, como análisis de datos, que incluyen: Escaneo, búsquedas predecesoras por lotes, fusión multi vía y fraccionamiento. Para cada algoritmo, examinaron el rendimiento utilizando independientemente CPU y GPU, así como utilizar de manera híbrida CPU y GPU.

Los principales resultados que encontraron son: Los algoritmos con accesos aleatorios a la memoria, que requieren de capacidad de cómputo y con poca contención de datos en memoria, mejoran el rendimiento si se hace uso de la GPU. Algoritmos que recuperan datos mediante acceso lineal a la memoria, como el escaneo, tienen un mejor rendimiento en la CPU que en la GPU. Finalmente, los algoritmos que trabajan sobre bajas contenciones de datos en memoria, el enfoque híbrido presenta mejor rendimiento, dado que el proceso en la GPU realiza el proceso sin afectar sustancialmente el proceso efectuado en la CPU.

Li H et al. (2019), analizan el procesamiento de consultas de un sistema de base de datos, desde una perspectiva de consultas concurrentes basados en unidades de procesamiento gráfico. Primero resaltan la gran capacidad de cómputo de las tarjetas GPU, comparando los rendimientos a nivel de *teraflops* - TFLOPS (que mide cuántos miles de millones de operaciones de punto flotante realiza un procesador en un segundo), de las CPU y las GPU. Las GPU generalmente van por encima de 10 TFLOPS, mientras que las CPU van por debajo de 1 TFLOPS.

En el trabajo, los autores proponen un modelo que es una variación multidimensional del problema de la mochila, para maximizar la concurrencia en un contexto multinúcleo. Presentan un modelo implementado con un algoritmo basado en programación dinámica. El algoritmo encuentra soluciones óptimas considerando concurrencia de subprocesos, con una complejidad seudo - polinomial.

La parte experimental se realizó utilizando tarjetas GPU de NVIDIA, donde se configuró los parámetros en función a los núcleos CUDA de las tarjetas, asignando tareas a cada núcleo para su ejecución simultánea y concurrente; habiendo logrado maximizar el rendimiento del sistema.

Hu et al. (2020), en su trabajo de investigación de optimización de consultas de composición múltiple utilizando tarjetas de procesamiento gráfico (GPU), parten de la premisa de que las consultas de bases de datos que incluyen composiciones múltiples son operaciones relevantes en los sistemas de gestión de bases de datos, por tanto, optimizar estas operaciones es un factor importante.

Este trabajo plantea aplicar una arquitectura paralela basada en las tarjetas GPU, en la intención de optimizar las composiciones múltiples en las consultas de bases de datos. Establece un método que considera un árbol de composición de costo mínimo en la unidad de procesamiento gráfico, aprovechando la capacidad de ejecución paralela que proporciona este

tipo de unidades de procesamiento. Se logra esta optimización sobre todo en consultas de composición complejas y que requieran alta carga de datos.

Se consideran diferentes posibilidades de arquitecturas paralelas basadas en el uso de las tarjetas GPU. Por ejemplo, utilizar sólo los múltiples nodos de procesamiento de una única tarjeta GPU o múltiples máquinas cada una con múltiples núcleos GPU. Como herramientas de desarrollo en una máquina se usó OpenMP considerando principalmente su compatibilidad con CUDA para articular los procesos entre la CPU y la GPU. Mientras que en múltiples máquinas se utilizó MPI para distribuir las tareas de composición múltiple a diferentes máquinas con sus respectivas GPU.

Lee et al. (2021) publicaron un artículo en el que plasman experiencias y consejos de balancear múltiples factores involucrados en la implementación de productos de bases de datos haciendo uso de arquitecturas híbridas o heterogéneas, con son el uso de simultáneo de capacidad de cómputo de CPU y GPGPU. Se presenta un estudio integral de productos de la industria que hacen uso de la arquitectura híbrida CPU/GPU a la que denominan RateupDB. Abordan problemas relevantes como la compensación entre rendimiento y productividad, considerando aplicaciones de tipo OLTP y OLAP.

Basan su trabajo en la perspectiva de que las tecnologías emergentes propician cambios drásticos en las diferentes estructuras de la sociedad; conceptos como informática móvil, intercambio de contenido digital y demás transformaciones dadas tanto en el campo económico, educativo y sociocultural, hace que las aplicaciones de bases de datos del mundo real requieren mayores funcionalidades que van más allá de las ofrecidas por los sistemas convencionales. Resumen estos nuevos requisitos en tres categorías: Necesidad de información en tiempo real, Potencia informática extrema y gestión integral de datos. Respecto al primer requisito, en muchos contextos empresariales, se requiere de información comercial en línea, que permitan efectuar análisis de datos y tomar decisiones en el momento oportuno. Respecto

al segundo requisito, las organizaciones cuentan cada vez más con grandes volúmenes de datos, que deben ser procesados en línea y obtener ventaja competitiva a partir de éstos. Respecto al tercer requisito, se plantean que los sistemas de bases de datos deben ser integrales, procesando la información de la organización de manera articulada. Estos aspectos inciden a la necesidad de la implementación de bases de datos con arquitectura híbrida CPU/GPU.

El artículo analiza diferentes partes que puedan ser optimizados en las aplicaciones de bases de datos, si se utiliza arquitectura híbrida CPU/GPU; orientándolas sobre todo a cargas de trabajo que requieran procesos de alto rendimiento, tanto a nivel de OLTP y OLAP.

Aranda (2016) en su tesis doctoral da una visión de los fundamentos teóricos e implementa bases de datos deductiva, extendiendo la funcionalidad del SQL de las bases de datos relacionales, agregando predicados lógicos que efectúan el proceso de deducción en base a la capacidad de recursión presente en las últimas versiones de los sistemas de gestión de bases de datos relacionales. El autor sintetiza su trabajo en la implementación del sistema HR-SQL, que está constituido por un lenguaje basado en la nomenclatura relacional, con funcionalidades deductivas, basadas en la recursión lineal y la recursión mutua.

Sin embargo, el trabajo es de corte eminentemente académico y queda sólo como una propuesta a ser estudiada para aplicarla al campo práctico. Por otra parte, no considera la posibilidad de utilizar algoritmos paralelos que permitan optimizar tiempos y permitir su aplicación en software de gestión.

1.5 Justificación

Justificación social

Como se manifestó en el planteamiento del problema, la tecnología hardware avanza vertiginosamente, dotando en la actualidad a los comunes equipos informáticos capacidades de cómputo heterogéneo, soportado en procesadores con varios núcleos CPU y/o núcleos GPGPU.

El problema es que estas capacidades de cómputo normalmente son ignoradas y desperdiciadas, sobre todo en el software de gestión administrativa.

Este trabajo es importante desde la perspectiva social, porque propende apoyar tecnológicamente a las organizaciones privadas y públicas, a aprovechar la tecnología hardware existente, para potenciar la funcionalidad el software convencional de gestión administrativa; que coadyuve al desarrollo socio económico de la colectividad.

Justificación académica

En la actual era del conocimiento, las diferentes disciplinas académicas son altamente dinámicas; más aún, las referidas a las Tecnologías de la Información y la comunicación. Se asume que el conocimiento científico crece vertiginosamente, tal como se puede apreciar en la "Curva de duplicación del conocimiento" propuesto por Richard Buckminster Fuller y referenciado en Unicentro Cúcuta (2019); quién proyecta que en los tiempos en que vivimos, el conocimiento se duplicará en cuestión de días.

En el ámbito universitario, la enseñanza de la algorítmica pareciera mantenerse en un estado anquilosante, circunscribiéndose prioritariamente al paradigma secuencial, dándose a entender implícitamente que la implementación de software obedece sólo a este paradigma. Si bien, se consideran aspectos de computación concurrente y paralela, esto se realiza en asignaturas avanzadas o de especialidad orientadas a investigación.

El contexto actual del hardware cuenta con alto potencial de cómputo, expresados en computadoras con más de un procesador y de naturaleza heterogénea, tal las multinúcleo y las GPGPU. Estas altas capacidades de cómputo generalmente son ignoradas, sobre todo en el desarrollo de software convencional de gestión; enfocándose sólo al desarrollo de aplicaciones técnicas o científicas.

Este trabajo es relevante, porque propone el uso de todo el potencial de cómputo del hardware actual, también en la implementación de software de gestión; por ende, en la de

cualquier tipo de software. Se espera que este hecho, implique la adecuación y reestructuración de los currículos de estudio, enfatizando el paradigma de computación paralela y heterogénea, desde asignaturas básicas.

Justificación tecnológica

Hoy en día las tecnologías de la información están cambiando la estructura funcional de las empresas, en especial la inteligencia artificial. Éstas ven la necesidad de incorporar innovaciones tecnológicas en sus procesos. En este contexto, se identifica dos tipos de procesos: Determinísticos y estocásticos. Se pone de manifiesto el auge en las tecnologías y herramientas de tipo estocástico como "Data mining". Sin embargo, también hay problemas que requieren de soluciones determinísticas; como, la automatización de procesos con una mínima o nula intervención humana.

Considero que, desde la perspectiva tecnológica, el presente proyecto es importante porque, coadyuvará a la aplicación de un nuevo paradigma en el desarrollo de aplicaciones de gestión administrativa, que permitirá:

- Potenciar los sistemas de información de tipo administrativo más allá del simple procesamiento de datos operacionales y análisis de información; sino que, en tiempo real permita deducir y gestionar información implícita (oculta o no visible) de la organización, que en el general de los casos es ignorada y desperdiciada.
- Potenciar la capacidad de cómputo del hardware existente, promoviendo el uso de los recursos con que cuenta. La computación paralela y las arquitecturas heterogéneas existentes propenderán a optimizar los tiempos de procesamiento de grandes volúmenes de datos; permitiendo así, darle valor agregado al software de gestión. Por ejemplo, efectuar procesos de automatización complementarios sin intervención humana, identificar patrones de comportamiento de los clientes en línea, efectuar proyecciones o predicciones en línea, etc.

1.6 Limitaciones

El presente trabajo, se orienta a la propuesta de un modelo de un motor de inferencia paralelo en arquitectura heterogénea, en base al cual, se efectúa el diseño e implementación de un prototipo; utilizando completamente algoritmos de tipo determinístico. No es intención la implementación de algún sistema experto específico. Se limita a motores de inferencia que complementen a las bases de datos transaccionales extendiendo su funcionalidad a la deducción de información implícita; enmarcadas dentro de un contexto de uso hacia el software aplicativo de gestión administrativa.

La implementación de los algoritmos paralelos en los procesos de inferencia está limitada al uso del hardware que generalmente se tiene en las organizaciones; es decir, computadoras con varios núcleos CPU, la tarjeta gráfica integrada (iGPU) y tarjetas gráficas dedicadas (dGPU), programadas con OpenCL. No se considera en la programación lenguajes específicos como CUDA, que sólo permiten la programación de tarjetas gráficas de NVIDIA.

El motor de inferencia implementado se circunscribe a efectuar procesos determinísticos de inferencia en software de gestión como contabilidad, ventas, Etc.

1.7 Objetivos

- ***Objetivo General***

Optimizar la deducción de información implícita en las bases de datos, mediante el uso de un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU).

- ***Objetivos específicos***

1. Determinar las características que debe tener un motor de inferencia paralelo en una arquitectura heterogénea, que permita optimizar la deducción de información implícita en las bases de datos.

2. Disminuir el tiempo de la optimización en la deducción de información implícita en las bases de datos, mediante el uso de un motor de inferencia paralelo en una arquitectura heterogénea.
3. Incrementar la eficiencia de la optimización en la deducción de información implícita en las bases de datos, mediante el uso de un motor de inferencia paralelo en una arquitectura heterogénea.
4. Proyectar los tiempos de procesamiento para cualquier volumen de datos, en la deducción de información implícita en las bases de datos, mediante el uso de un motor de inferencia paralelo en una arquitectura heterogénea.

1.8 Hipótesis

- ***Hipótesis General***

Si se usa un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU) se optimiza la deducción de información implícita en las bases de datos.

- ***Hipótesis específicas***

1. Si se determinan las características relevantes que debe tener un motor de inferencia paralelo en una arquitectura heterogénea, éstas permitirán optimizar la deducción de información implícita en las bases de datos.
2. Si se usa un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU), se disminuye el tiempo de la optimización en la deducción de información implícita en las bases de datos.
3. Si se usa un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU), se incrementa la eficiencia de la optimización en la deducción de información implícita en las bases de datos.

4. Si se usa un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU), se proyecta los tiempos de procesamiento para cualquier volumen de datos, en la deducción de información implícita en las bases de datos.

II. MARCO TEÓRICO

2.1 Marco Conceptual

2.1.1 Lógica

El concepto de lógica es bastante amplio; la intención del presente trabajo es utilizar el concepto de lógica circunscrito a la lógica formal matemática, más específicamente a la lógica proposicional, que coadyuvará a la definición de las reglas que permitirán implementar procesos de inferencia. En este sentido, se considera la definición dada por el Diccionario de la Real Academia Española (DRAE, 2023), en la que define la Lógica Formal y Matemática como: "Lógica que opera utilizando un lenguaje simbólico abstracto para representar la estructura básica de un sistema".

2.1.1.1 ***Lógica Proposicional.*** Es una rama de la lógica matemática y se constituye en un sistema formal diseñado para analizar proposiciones vinculadas con conectores lógicos. La acepción del término proposición se debe conceptualizar desde la perspectiva matemática; así, recurriendo al DRAE (2023) se tiene que proposición es "Enunciación de una verdad demostrada o que se trata de demostrar". También se debe considerar que vinculando las proposiciones mediante los operadores o conectores lógicos se puede formar otras proposiciones de mayor complejidad.

La lógica proposicional permite formular sistemas lógicos, de modo que, con un número finito de pasos se puede determinar si cada proposición es verdadera o falsa; en base a un léxico, sintaxis y semántica; lo que, le permite determinar si un sistema lógico es verdadero o falso. Este hecho, le da el carácter de completa a la lógica proposicional.

Léxico

Está constituido por: Letras proposicionales, que representan una proposición que puede ser verdadera o falsa. Por ejemplo: p, q, r, s, t, etc. Símbolos lógicos, dentro de las que se tiene constantes lógicas como verdadero y falso, conectivas unarias como negación y

operadores lógicos o conectivas binarias como la conjunción, disyunción, la implicación, etc. Finalmente, los símbolos auxiliares como los paréntesis que permiten determinar la procedencia de las operaciones evitando así la ambigüedad. Este léxico se resume en la figura 6.

Figura 6

Resumen del léxico de la lógica proposicional

Símbolo	Significado	Tipo/Operación	Alternativas
T	verdadero	booleano	true, 1, ⊤
F	falso	booleano	false, 0, ⊥
p	p	afirmación	P, p_1, cosa
¬p	no p	negación	~p, p', $\overline{p}$
p ∧ q	p y q	conjunción	p & q
p ∨ q	p o q	disyunción	p | q
p → q	si p entonces q	condicional, implicación	p - q
p ↔ q	p si y sólo si q	bicondicional, equivalencia	p ≡ q, p ⇔ q
(p)	p	precedencia	[p]

Nota. Información tomada de Russell (2022, p 235).

Sintaxis

Determina como se deben estructurar las sentencias utilizando el léxico de la lógica proposicional. Un resumen se muestra en la figura 7.

Figura 7

Resumen de la sintaxis de la lógica proposicional

Sentencias atómicas
(Fórmula atómica o literal)
Es una única letra proposicional o una constante lógica (T, F)
- Literal positivo: afirmación (p)
- Literal negativo: negación (¬p)

Sentencias complejas
(Sentencia / Fórmula)
- Una fórmula atómica: T, F, p, q, r, ...
- Una negación de una fórmula: ¬p, ¬¬q, ¬T, ¬(p ⇒ q), ¬[r ∧ (q ∨ ¬p)]
- Una conjunción, disyunción, condicional o bicondicional de dos fórmulas: r ∧ q, r ∧ (q ∨ ¬p), r ⇒(p ⇒ q)

Implicación
Regla o sentencia ***Si-Entonces***
p ⇒ q
- Premisa / antecedente: Parte izquierda (p)
- Conclusión / consecuente: Parte derecha (q)

Precedencia
De mayor a menor:
- Negación (¬)
- Conjunción (∧)
- Disyunción (∨)
- Condicional
- Bicondicional

Se evalúa de izquierda a derecha. Alterado por paréntesis.

Nota. Resumen elaborado en base a Russell (2022, p 235, 236, 237).

Semántica

La semántica define la interpretación que se debe dar a cada elemento de la lógica proposicional, para determinar el valor de verdad de una sentencia respecto a un modelo en concreto. Se ilustra en la figura 8.

Figura 8

Semántica de los operadores lógicos de la lógica proposicional

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
falso	*falso*	*verdadero*	*falso*	*falso*	*verdadero*	*verdadero*
falso	*verdadero*	*verdadero*	*falso*	*verdadero*	*verdadero*	*falso*
verdadero	*falso*	*falso*	*falso*	*verdadero*	*falso*	*falso*
verdadero	*verdadero*	*falso*	*verdadero*	*verdadero*	*verdadero*	*verdadero*

Nota. Russell (2022, p 237).

2.1.1.2 Métodos de Inferencia. La lógica proposicional permite implementar diferentes métodos o patrones de inferencia. Desde una perspectiva genérica, se puede mencionar los siguientes:

Tablas de verdad.

Deducción con reglas de inferencia.

Resolución.

Encadenamiento hacia delante.

Encadenamiento hacia atrás.

Tableaux

Algunos de estos métodos son utilizados por los lenguajes de programación lógicos como el PROLOG, que utiliza el método de inferencia de “Encadenamiento hacia atrás”. Mientras que, otros métodos son más apropiados para procesos manuales. Para propósitos del presente trabajo, se utilizará los conceptos asociados al método de resolución; la justificación del porqué se realizará en los siguientes acápites.

2.1.1.3 Método de Inferencia por Resolución. El método de inferencia por resolución es uno de los métodos que se ejecuta de manera rápida y cuya implementación es sencilla; debido principalmente, a que exige que previamente la sentencia de lógica de proposiciones sea expresada en la forma normal clausulada. A su vez la forma normal clausulada requiere que previamente la sentencia de lógica de proposiciones sea transformada a la forma normal conjuntiva o disyuntiva.

Una vez que la sentencia de lógica proposicional esté en cualquiera de las formas clausuladas, el método de resolución implica utilizar dos reglas de resolución:

Regla I de resolución. Si en una sentencia de lógica proposicional en forma normal clausulada, se tiene un literal y su negación en dos cláusulas distintas, entonces, se puede eliminar dicho literal uniendo el resto de los literales de ambas cláusulas en una única cláusula.

Regla II de resolución. Si en una sentencia de lógica proposicional en forma normal clausulada se tiene un literal y su negación en la misma cláusula, entonces, se puede eliminar dicha cláusula. La fundamentación para esta regla es qué, si se tiene un literal y su negación en la misma cláusula, esta cláusula es verdadera, independientemente del valor de los otros literales.

Finalmente, el método de inferencia por resolución consiste en utilizar la demostración por contradicción; para lo cual, se agrega la negación de la sentencia que se desea inferir a la sentencia de lógica proposicional clausulada; luego, se aplica de manera reiterada las dos reglas de resolución enunciadas previamente según corresponda. Si se llega al final a una cláusula vacía, entonces la sentencia agregada se puede inferir de la sentencia de lógica proposicional; caso contrario, se concluye que no se puede inferir.

Desde la perspectiva del uso de arquitecturas heterogéneas, sobre todo del uso de GPGPU, el método más adecuado para los procesos de inferencia es el método de “Resolución”, porque utiliza formas normales que simplifican la representación de sistemas

lógicos. Otra razón fundamental es que la arquitectura GPGPU no permite el uso de la recursión; en consecuencia, los otros métodos de inferencia no son pertinentes, porque los algoritmos que subyacen en ellos requieren de la recursión; mientras que el método de resolución no requiere de procesos o evaluaciones recursivas.

2.1.1.3.1 Forma Normal Conjuntiva (FNC). Un aspecto interesante de la lógica de proposiciones es que provee de un conjunto de sentencias equivalentes, en virtud de las cuales se puede transformar cualquier sentencia de lógica de proposicional a otras equivalentes, incluso a formas normales conjuntivas o disyuntivas.

Tal como se indica en Russell (2022, p 244), se dice que una sentencia de lógica proposicional está en forma normal conjuntiva si corresponde a una conjunción de cláusulas, donde a su vez, una cláusula es una disyunción de literales.

A continuación, en la figura 9 se muestra una tabla con un mínimo de equivalencias requeridas para poner cualquier sentencia de lógica proposicional en forma normal conjuntiva.

Figura 9

Resumen de equivalencias de la lógica proposicional

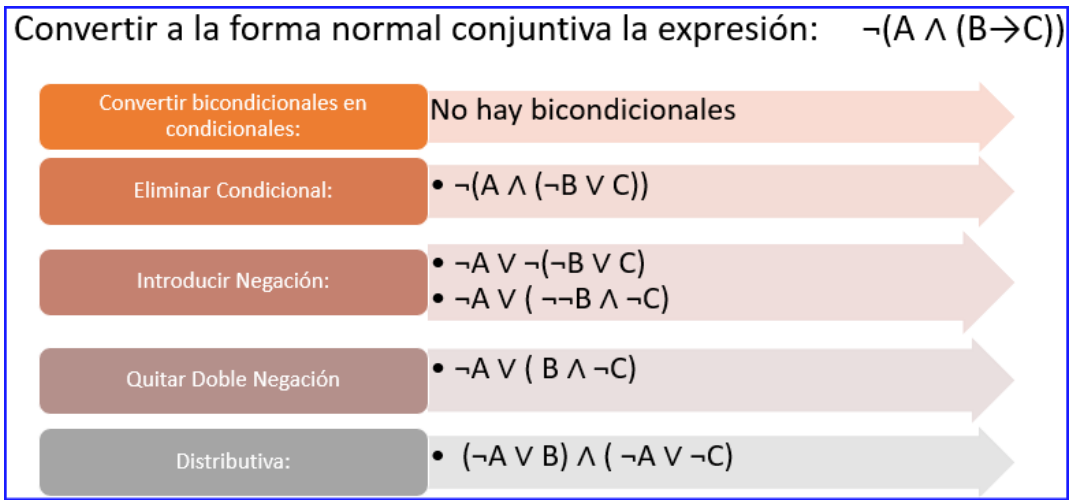
1. Convertir bicondicionales en condicionales:
   $A \leftrightarrow B \equiv (A \rightarrow B) \land (B \rightarrow A)$
2. Convertir condicionales en disyunciones:
   $A \rightarrow B \equiv \neg A \lor B$
3. Introducir negaciones:
   $\neg(A \land B) \equiv (\neg A \lor \neg B)$
   $\neg(A \lor B) \equiv (\neg A \land \neg B)$
4. Eliminar dobles negaciones:
   $\neg\neg A \equiv A$
5. Aplicar distributiva:
   $A \lor (B \land C) \equiv (A \lor B) \land (A \lor C)$
   $(A \lor B) \land C \equiv (A \lor C) \land (B \lor C)$

Nota. Russell (2022, p 245).

Haciendo uso de estas equivalencias se puede transformar una sentencia de lógica proposicional a la forma normal conjuntiva, tal como se puede ilustra en la figura 10.

Figura 10

Conversión a la forma normal conjuntiva



Nota. Se utilizó los pasos detallados en Russell (2022, p 245).

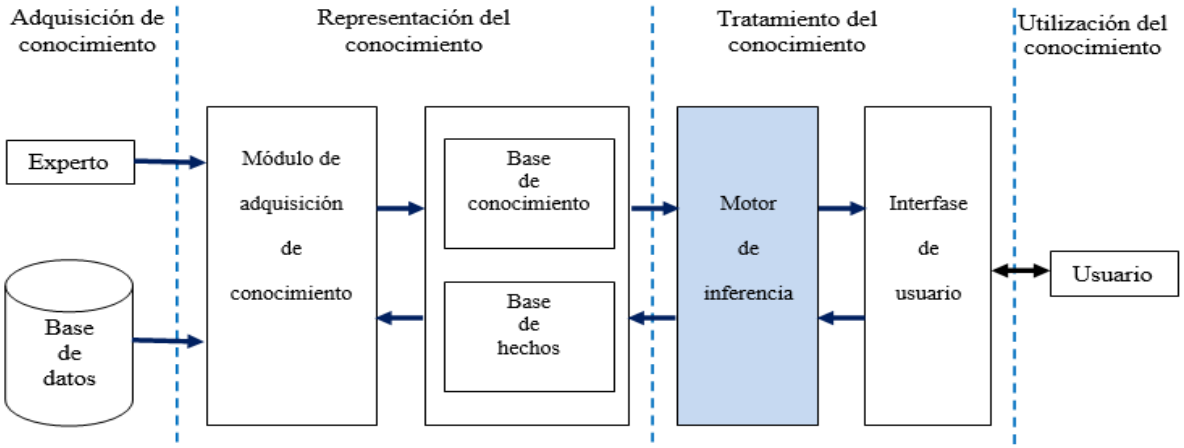
La propuesta del presente trabajo es transformar las diferentes reglas de la base de conocimientos a la forma normal conjuntiva; porque, ésta es la más apropiada para la implementación del paralelismo del tipo SIMT (single instruction - multiple threads), que es la más pertinente en la arquitectura GPGPU; de modo que se pueda explotar todo el potencial de cómputo que ofrece esta arquitectura.

2.1.2 Motor de Inferencia

El motor de inferencia es el componente que implementa el algoritmo de la lógica de inferencia o razonamiento, que se efectúa sobre los hechos y las reglas de la base de conocimientos. Es el componente que también, activa y controla el proceso de inferencia. Fundamentalmente, se encarga de gestionar el conocimiento explícito existente, para luego, a partir de las reglas de deducción, explicitar conocimientos implícitos que subyacen en los datos o conocimientos previos. Actúa como el medio de interacción entre el usuario y la base de conocimientos; constituyéndose, en el principal componente de los sistemas expertos. La figura 11 esquematiza la función del motor de inferencia en un sistema experto.

Figura 11

Componentes de un sistema experto



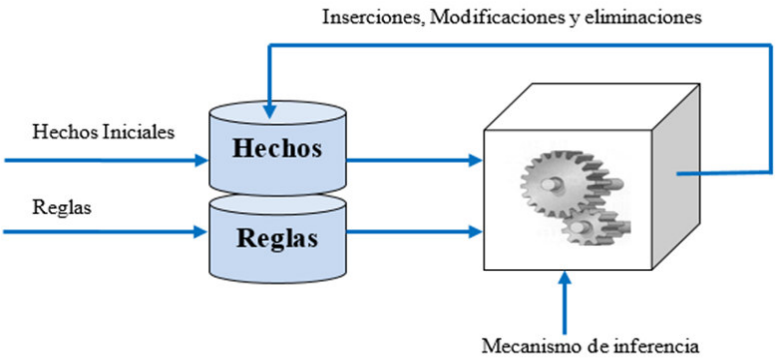
Nota. Elaboración propia.

Sin embargo, un motor de inferencia se puede aplicar también, sobre bases de datos convencionales; convirtiéndolos de esta manera, en bases de datos deductivas. Es a esta funcionalidad que se orienta el presente trabajo.

2.1.2.1 Arquitectura de un Motor de Inferencia. El proceso de inferencia está constituido principalmente por tres componentes: El conocimiento fáctico, denominado hechos; las condicionantes de deducción, denominada reglas; el algoritmo que interpreta las reglas y las aplica sobre los hechos. Se ilustra en la figura 12.

Figura 12

Componentes de un proceso de inferencia



Nota. Elaboración propia.

2.1.3 Arquitecturas Heterogéneas

Desde la propuesta por John Von Neumann en 1945, de las computadoras monoprocesador de ejecución secuencial, la tecnología ha tenido avances significativos en materia de hardware. Hoy en día, con el propósito de optimizar el rendimiento computacional se cuenta con arquitecturas heterogéneas, que son infraestructuras de cómputo que utilizan más de un tipo de procesador. En la actualidad es normal que los equipos de cómputo tengan procesadores multinúcleo y tarjetas GPGPU (integradas - iGPU o dedicadas - dGPU). Sin embargo, también existen equipos que adicionalmente cuentan con otros tipos de procesadores; tal las FPGA, cuya referencia se puede ver en BBC News (2021); y las ASIC, cuyo detalle se puede ver en Intel - ASIC (2021).

2.1.3.1 Taxonomía de Arquitecturas Heterogéneas. Se tienen clasificaciones convencionales de las arquitecturas computacionales, tal como lo manifiesta Patterson y Hennessy (2014); sin embargo, en la figura 13 se propone una adecuación que considera sólo los procesadores del tipo CPU, Multinúcleo y GPU; en función de las cuales, se indica también el tipo de procesador pertinente para cada tipo de arquitectura.

Figura 13

Taxonomía de arquitecturas heterogéneas

		DATOS	
		Single (Único)	Multiple (Varios)
INSTRUCCIONES	Single (Único)	SISD Single Instruction - Single Data (Programación convencional en un CPU)	SIMD Single Instruction - Multiple Data (Programación en Núcleos y GPU)
	Multiple (Varios)	MISD Multiple Instruction - Single Data (Programación en Núcleos)	MIMD Multiple Instruction - Multiple Data (Programación en Núcleos)

Nota. Elaboración propia.

- SISD (single instruction - single data) que consta de un procesador en el que se ejecuta una única secuencia de instrucciones que procesa un único conjunto de datos.
- SIMD (single instruction - multiple data) que también opera con un procesador, que permiten simultáneamente ejecutar operaciones sobre múltiples conjuntos de datos. Las GPGPU implican una nueva arquitectura de cómputo, como es la SIMT (single instruction - multiple threads), que permite que una única instrucción sea ejecutada por diferentes hilos procesando múltiples datos.
- MISD (multiple instruction - single data) que considera la posibilidad de varias unidades de proceso que operen sobre un único conjunto de datos.
- MIMD (multiple instruction - multiple data) que opera con varios procesadores efectuando procesos sobre diferentes conjuntos de datos. Los equipos de cómputo multinúcleo tienen un funcionamiento acorde a este tipo de arquitectura.

2.1.3.2 Tipos de Procesadores en las Arquitecturas Heterogéneas. En este punto se va a revisar sólo los procesadores vigentes en las arquitecturas heterogéneas. Se asume, que toda arquitectura como mínimo debe contar con un procesador CPU de un solo núcleo; adicionalmente, se puede contar con otros tipos de procesadores, orientados a optimizar el rendimiento computacional del equipo.

2.1.3.2.1 CPU Multinúcleo. En los actuales procesadores, en una sola unidad física se tienen varios núcleos, éstos funcionan como unidades de procesamiento completos y ejecutan instrucciones de manera autónoma, permitiendo incluso la simultaneidad.

Se hace referencia al multinúcleo en la intención de que se utilizará está arquitectura en la evaluación del rendimiento de los algoritmos paralelos subsumidos en los motores de inferencia. Hay abundante información sobre multinúcleo, por ejemplo, en la página web TechLib (2021).

Un primer nivel de computación heterogénea se da con el uso de CPU con dos o más núcleos de procesamiento. Básicamente son procesadores independientes integrados dentro de un solo chip, que pueden realizar procesos autónomos. La principal ventaja de este tipo de procesadores es que aumentan significativamente el rendimiento computacional, aunque no en la proporción del número de núcleos.

Esta tecnología ha sido posible, gracias a la construcción de semiconductores de 90 nanómetros, que posibilitaba la integración de más transistores en espacios reducidos; dando lugar a los procesadores conocidos como dual-core. Se estima que, en los inicios de esta tecnología, se podía integrar aproximadamente 230 millones de transistores.

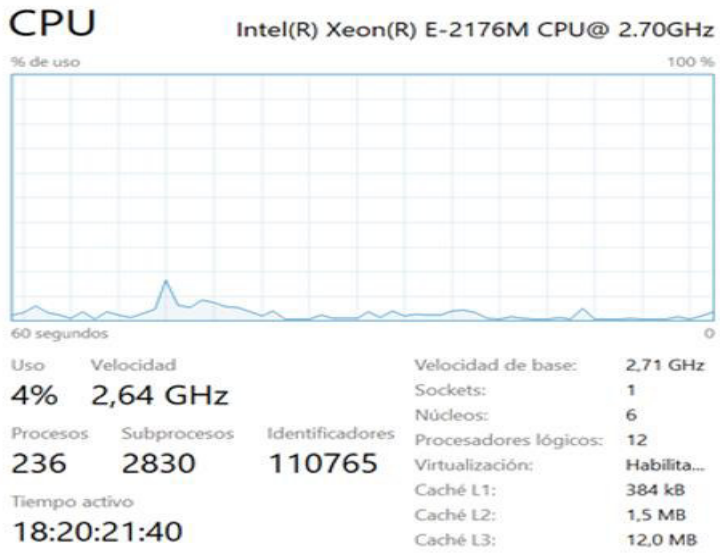
Comparando este tipo de procesadores con los sistemas con múltiples procesadores, resalta como una ventaja sustancial, el menor consumo eléctrico. También es relevante, que el espacio que ocupan es mucho menor; también los sistemas monoprocesadores respecto a los procesadores multinúcleo disipan más calor, que obviamente van a requerir de fuentes de alimentación más especializadas; en consecuencia, su costo también será mayor.

La tendencia a lo largo de los años ha sido integrar más núcleos en el mismo chip. Es así, por ejemplo, que Intel lanzó progresivamente al mercado sus procesadores multinúcleo i3, i5, i7 e i9. Cada vez se tiene procesadores con más núcleos, tal como manifiesta Intel (2019) el procesador *Intel core i9-10980X Extreme Edition* de 18 núcleos. El mismo fenómeno se da en las Workstation, los servidores y las arquitecturas HPC, con microprocesadores Intel Xeon, tal como indica Intel Xeon (2019) el procesador *Intel Xeon Platinum 9200* de 56 núcleos.

Finalmente, en este tipo de procesadores, está también generalizado el uso de procesadores virtuales, de manera que con cada núcleo se puede atender dos hilos de procesamiento; en consecuencia, asociado a cada núcleo físico, se tiene dos núcleos lógicos. Una ilustración de la información de los núcleos, se tiene en la figura 14.

Figura 14

Información de los núcleos físicos y lógicos



Nota. - Obtenido de la pestaña rendimiento del administrador de tareas de Windows.

2.1.3.2.2 GPGPU. En sus inicios las CPU de las computadoras personales presentaban problemas en el pintado de los pixeles de las pantallas gráficas, porque éstas involucraban muchas veces cálculos en aritmética de punto flotante, para determinar los colores de cada píxel; la tecnología suplió esta limitación, agregando a los CPU coprocesadores matemáticos, cuya única función consistía en acelerar los cálculos de aritmética de punto flotante, requeridos en la manipulación de las pantallas gráficas. Luego, ante los requerimientos de procesamiento gráfico cada vez más exigentes, los coprocesadores matemáticos evolucionaron hasta convertirse en tarjetas individuales o independientes, posibilitando la creación de chips más grandes, con mayor número de transistores y circuitos digitales; funcionando sobre conexiones con menores consumos energéticos. Este proceso de evolución fue liderado por NVIDIA, tal como se puede apreciar en la historia de NVIDIA (2020), quienes en el año 1999 crearon las denominadas “unidades de procesamiento gráfico” (GPU), dedicados exclusivamente al procesamiento gráfico, con el objetivo de liberar la carga de

trabajo a la CPU, sobre todo en los videojuegos. Muchos fabricantes empezaron así con una carrera de ofrecer tarjetas gráficas cada vez con mejores características y capacidades. NVIDIA en su posición de empresa líder en este tipo de tecnología, ponderó la posibilidad de utilizar las GPU para efectuar cálculos matemáticos no sólo orientado a prestaciones gráficas, sino a efectuar procesos de propósito general, dando lugar así a las "Unidades de procesamiento gráfico de propósito general" GPGPU (general-purpose computing on graphics processing units).

No es propósito del presente trabajo, describir las características técnicas ni electrónicas de los GPU, sino, enfatizar la gran posibilidad que ofrecen para la implementación de aplicaciones con algorítmica paralela que pueden funcionar sobre esta arquitectura. De hecho, hoy en día ya existen muchas aplicaciones funcionando sobre arquitecturas CPU + GPU, aplicaciones en el ámbito económico y financiero, gestión de grandes bases de datos, simulaciones físicas, bioinformática, inteligencia artificial, aprendizaje automático de máquinas, etc.

También se debe enfatizar el hecho de que las computadoras de los hogares y de las empresas incluso más pequeñas, ya cuentan con esta capacidad de computación heterogénea; pero lamentablemente, por desconocimiento se subutilizan estos equipos, explotándolos sólo como máquinas con un único procesador CPU y se ignora la capacidad GPGPU.

El reto es propiciar un nuevo paradigma de desarrollo de aplicaciones que optimicen el uso de toda la infraestructura instalada en los equipos de cómputo, enfocado principalmente a mejorar los tiempos de proceso en entornos de interacción en tiempo real. Por ejemplo, muchos cálculos numéricos, tal la solución de grandes sistemas de ecuaciones lineales y no lineales utilizados en diferentes ingenierías, reducir sus tiempos de ejecución de varios minutos a segundos.

Modelos de arquitectura GPGPU

Básicamente, las GPGPU se presentan en dos arquitecturas: En tarjetas integradas (iGPU) y en tarjetas dedicadas (dGPU). Los equipos de cómputo pueden venir equipadas sólo con tarjetas iGPU, sólo con tarjetas dGPU o con ambas.

Arquitectura iGPU

Son tarjetas gráficas integradas en el procesador central del computador. La principal ventaja de este tipo de tarjetas es que su costo es bajo; pero, en la misma medida el rendimiento también es bajo, porque comparten el mismo ancho de banda del CPU así como la memoria RAM.

Los actuales computadores personales, generalmente ya vienen equipadas con este tipo de tarjetas, en las que se pueden ejecutar aplicaciones de algorítmica paralela básicas.

Arquitectura dGPU

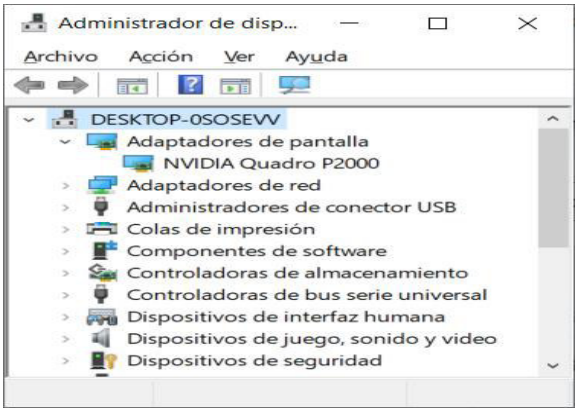
Una dGPU es una unidad de procesamiento gráfico que se encuentra en una tarjeta gráfica independiente de la CPU. Generalmente se hace uso de este tipo de tarjetas, cuando se requiere de mayor rendimiento en la capacidad gráfica de las aplicaciones, procesamiento numérico intensivo, aplicaciones de inteligencia artificial, etc.

Algunos computadores pueden venir equipadas también con tarjetas gráficas dedicadas; es decir tarjetas gráficas independientes, que normalmente ofrecen mejores prestaciones que las tarjetas integradas.

El computador que se utilizó en el presente trabajo corresponde a una HP Z4 G4 Workstation, con un procesador Intel(R) Xeon(R) W-2123 CPU @ 3.60GHz 3.60 GHz. Éste viene equipado sólo con una tarjeta del tipo dGPU, porque está orientado a correr aplicaciones con algoritmos paralelos. El detalle se muestra en la figura 15.

Figura 15

Tarjeta gráfica dGPU de la HP Z4 G4 Workstation



Nota. Obtenido del adaptador de pantallas del administrador de dispositivos.

Como se puede apreciar, este computador cuenta sólo con la tarjeta del tipo dGPU NVIDIA Quadro P2000. Las especificaciones técnicas se pueden obtener de la página web del fabricante, en este caso NVIDIA (2018, p. 1), tal como se muestra en la figura 16.

Figura 16

Características técnicas de la NVIDIA QUADRO P2000

SPECIFICATIONS

GPU Memory	5 GB GDDR5
Memory Interface	160-bit
Memory Bandwidth	Up to 140 GB/s
NVIDIA CUDA® Cores	1024
System Interface	PCI Express 3.0 x16
Max Power Consumption	75 W
Thermal Solution	Active
Form Factor	4.4" H x 7.9" L, Single Slot
Display Connectors	4x DP 1.4
Max Simultaneous Displays	4 direct, 4 DP 1.4 Multi-Stream
Display Resolution	4x 4096x2160 @ 120Hz 4x 5120x2880 @ 60Hz
Graphics APIs	Shader Model 5.1, OpenGL 4.5[4], DirectX 12.0[5], Vulkan 1.0[4]
Compute APIs	CUDA, DirectCompute, OpenCL™

Nota. Obtenido de la página web de NVIDIA (2018)

2.1.3.2.3 FPGA. El incesante y dinámico avance de la tecnología de la información y comunicaciones, hace que ésta incursione en áreas muy especializadas, donde se requiere rendimientos computacionales altos. Esta situación, impulsó la posibilidad de embeber software en los chips; dado que los circuitos integrados diseñados para una funcionalidad predeterminada y específica son mucho más veloces.

Se inició entonces la etapa de los circuitos integrados con software embebido, dando lugar a dispositivos más evolucionados como los denominados FPGA (Field programable gate array), tal lo indicado por Rucci (2017, p.1).

Un FPGA es un circuito integrado, que permite programar su funcionalidad electrónica, a nivel de operaciones lógicas elementales. Consta de un conjunto de semiconductores, configurados en matrices de bloques lógicos sin conectar; pero, con una alta capacidad de conexión; permitiendo así, mucha flexibilidad al momento de programarlos para propósitos muy específicos. Básicamente, son dispositivos que permiten crear funcionalidades físicamente en el chip, mediante el cargado de scripts implementados en lenguajes específicos del tipo “Hardware Description Language”.

La principal ventaja de las FPGA es que estas son reprogramables, dando la posibilidad que se puede configurar un número ilimitado de circuitos sobre el FPGA. Actualmente se utiliza en proyectos de investigación que requieren de computadoras de alto rendimiento; tal los proyectos: aéreo espaciales, industria médica, etc. Para referencias técnicas se puede revisar Intel FPGAs Resource Center (2021).

El inconveniente que tienen las FPGA, es que, al ser reprogramables, son más lentas que los circuitos integrados con funcionalidad fija embebida.

2.1.3.2.4 ASIC. Dentro de los desafíos de las tecnologías de la información, hay problemas que necesitan de mayor rendimiento computacional que las proporcionadas por las FPGA; se requiere entonces de chips más optimizados. Entonces, surgen los circuitos

integrados con la funcionalidad de una aplicación específica y concreta, a las que se las denomina ASIC (Application-specific integrated circuit).

Con el incesante avance de la tecnología, también se dio un avance significativo en la industria, que permitió el inicio de la cuarta revolución industrial, tal como se puede ver en BBC News (2021); donde, se necesita optimizar y personalizar la funcionalidad de algunos equipos o maquinarias, inmersas en los procesos productivos de estos centros industriales. Es en estos contextos que los circuitos integrados de aplicación específica tienen especial relevancia.

Existen diferentes tipos de estos chips y para diferentes propósitos. Los fabricantes como Intel ofrecen una variedad de modelos, ver ASIC – Intel (2021) para una mayor referencia.

2.1.3.3 Lenguajes de Programación en las Arquitecturas Heterogéneas. En un inicio, los diferentes paradigmas de lenguajes de programación estaban orientados al modelo de cómputo de Von Neumann, específicamente al paradigma SISD de la taxonomía de Flynn, que corresponde a la arquitectura más básica de cómputo.

Con el surgimiento de las arquitecturas heterogéneas, también se hace necesario el surgimiento de nuevos paradigmas de lenguajes de programación. Así, para el caso de programar las GPGPU, surgen lenguajes como CUDA y OpenCL. Para la programación de los circuitos digitales, sean estos de tipo ASIC o FPGA, surgen los denominados lenguajes HDL (Lenguajes descriptores de hardware).

Por la orientación del presente trabajo, se revisarán sólo los lenguajes orientados a los GPGPU; considerando que, para la programación de los CPU multinúcleo, los lenguajes de programación actuales proporcionan las respectivas librerías.

2.1.3.3.1 CUDA. Tal como se puede apreciar en la página oficial de NVIDIA - CUDA (2021), CUDA (Compute Unified Device Architecture) es un nuevo paradigma de programación, orientado a arquitecturas de computación paralela, en dispositivos GPGPU fabricados por NVIDIA. Es un lenguaje que incluye un compilador, creado por la empresa NVIDIA; de modo que, se pueda implementar código de propósito general, aprovechando el motor de cálculo paralelo subyacente en este tipo de tarjetas gráficas; permitiendo así, mejorar el rendimiento computacional de una CPU, en la solución de problemas computacionales complejos.

La lógica de cómputo de CUDA obedece al modelo SIMD de la taxonomía de Flynn, más específicamente al modelo SITM, donde una instrucción simple es procesada por múltiples hilos, donde cada hilo se ejecuta en un núcleo GPU. Básicamente, se debe identificar e implementar el módulo que debe ejecutarse en paralelo, éste debe implementarse como uno módulo *kernel*, que luego es asignado a cada procesador de la GPU.

Desde una perspectiva muy general, el uso de la arquitectura heterogénea con CUDA, implica programar lógica para su ejecución tanto en la CPU (Denominada Host) y en la GPU (denominada Device). En consecuencia, el programa contempla instrucciones para ambos tipos de procesadores, de acuerdo con el siguiente esquema general:

- Declarar variables para el Host
- Inicializar las variables del host (Por ejemplo, arreglos)
- Declarar variables en el Device
- Reservar memoria en la GPU
- Copiar datos de la memoria de la CPU a la memoria de la GPU
- Dimensionar el kernel de la GPU
- ***Ejecutar el módulo del kernel en la GPU***
- Esperar a que la GPU finalice su proceso, antes de acceder al host

- Copiar datos (resultados) de la memoria de la GPU a la CPU
- Liberar memoria de la GPU
- Mostrar resultados

El código asociado a todas estas fases se ejecuta en la CPU, excepto la fase mostrada en negrita y cursiva, que es la que corresponde a la implementación de las instrucciones que deben ejecutarse en paralelo y en cada núcleo GPU. Normalmente, el código que debe ejecutarse en la GPU debe circunscribirse a la parte de la lógica del algoritmo que se puede ejecutar en paralelo. Por ejemplo, en la suma de dos vectores, la parte que puede ejecutarse en paralelo corresponde a la suma del i-ésimo elemento de los dos vectores, que será ejecutada en cada núcleo de la GPU; luego, si la GPU consta de núcleos suficientes (un número mayor o igual al número de elementos de los vectores); entonces la suma se ejecutará en todos los núcleos a la vez, en una unidad de tiempo. En consecuencia, el tiempo de ejecución será casi el mismo, para cualquier tamaño de los vectores. La figura 17 ilustra la lógica del algoritmo que se ejecutará en paralelo en cada uno de los miles de núcleos que pueda tener la GPU.

Figura 17

Función CUDA para sumar el i-ésimo elemento de dos vectores

CUDA: Ejemplo base

```
// -- Funcion kernel que se ejecutará en cada núcleo GPU (Device)
__global__ void add(int n, float *x, float *y, float *z)
{
        // -- Recuperar el identificador del hilo que se está ejecutando
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        // -- Ejecutar el cálculo asignado al hilo
        z[i] = (x[i] + y[i]);
}
```

Nota. Código elaborado por el autor.

Una de las ventajas de CUDA es la menor curva de aprendizaje, porque se basa en el lenguaje de programación C++; fortalecido con un conjunto de extensiones, siendo las más relevantes, la gestión jerarquizada de hilos, la gestión de memoria compartida y los aspectos de sincronización. También es importante resaltar, que ofrece interfaces que permiten integrarse con aplicaciones implementadas en otros lenguajes de programación, como FORTRAN, PYTHON, etc.

La mayor desventaja de CUDA es que sólo se puede programar algoritmos que correrán en tarjetas gráficas de NVIDIA; es decir, sólo se aplican a dispositivos del tipo dGPU de NVIDIA. No se puede utilizar para tarjetas dGPU de otros fabricantes, como AMD: menos aún en dispositivos de tipo iGPU.

2.1.3.3.2 OpenCL. Es también un nuevo paradigma de lenguajes de programación para arquitecturas heterogéneas. Es un estándar abierto que significa Open Computing Language, patrocinado por el grupo tecnológico Khronos – OpenCL (2018). Su ámbito de aplicación es la programación paralela en diversos dispositivos considerados como aceleradores, tal las CPU multinúcleo o las GPGPU. Tiene los mismos alcances que CUDA, con la ventaja de que se puede utilizar con cualquier tipo de dispositivo, incluso tarjetas gráficas NVIDIA. Al ser una plataforma abierta, puede utilizarse también en tarjetas de tipo iGPU y dGPU, incluso se puede potenciar el rendimiento computacional haciendo uso a la vez de ambos tipos de tarjetas.

La lógica de cómputo de OpenCL obedece también al modelo SIMD de la taxonomía de Flynn, más específicamente al modelo SITM, donde una instrucción simple es procesada por múltiples hilos.

Con OpenCL también se programa aplicaciones que se ejecutarán en una arquitectura heterogénea. El código debe contemplar instrucciones que se deban ejecutar en la CPU (Host)

y en la GPU (Device). El esquema general de una aplicación en OpenCL considera las siguientes fases:

- Inicializar los dispositivos CL disponibles
- Declarar las variables en el host
- Implementar o declarar el código del dispositivo, que se ejecutará en paralelo
- Compilar el código del dispositivo
- Declarar las variables en el Device
- Asignar como parámetros las variables de la memoria del host a la memoria del Device
- Definir el número de núcleos a utilizar
- ***Ejecutar el módulo en el dispositivo***
- Recuperar los resultados del Device al host y mostrar los resultados

La mayor parte del código se ejecuta en la CPU, excepto la fase mostrada en negrita y cursiva, que es la que corresponde a la implementación de las instrucciones que deben ejecutarse en paralelo en el dispositivo. La figura 18 ilustra la lógica del algoritmo que se ejecutará en paralelo en cada uno de los miles de núcleos que pueda tener la GPU.

Figura 18

Función OpenCL para sumar el i-ésimo elemento de dos vectores

CUDA: Ejemplo base

```
// -- Funcion kernel que se ejecutará en cada núcleo GPU (Device)
__global__ void add(int n, float *x, float *y, float *z)
{
        // -- Recuperar el identificador del hilo que se está ejecutando
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        // -- Ejecutar el cálculo asignado al hilo
        z[i] = (x[i] + y[i]);
}
```

Nota. Código elaborado por el autor.

El lenguaje OpenCL también presenta la ventaja de tener una menor curva de aprendizaje, porque también se basa en el lenguaje de programación C++; fortalecido con un conjunto de extensiones; dentro de las que se resalta la gestión de hilos, la gestión de memoria compartida y los aspectos de sincronización. También cuenta con implementaciones de API's que permiten interactuar con aplicaciones implementadas en otros lenguajes de programación, como C Sharp, PYTHON, etc. Otro aspecto relevante, es que es abierto y gratuito, se ejecuta en múltiples plataformas, encapsula las características específicas del hardware, haciéndolas transparentes al momento de la implementación de aplicaciones.

OpenCL también presenta algunas desventajas, dentro de las que podemos citar, que OpenCL implementa una interfaz uniforme para los diferentes tipos de dispositivos, lo que implica que siempre presentará una eficiencia algo menor respecto a los lenguajes específicos como CUDA. También que requiere aún del desarrollo de más librerías y ejemplos suficientemente documentados, orientados a diversas aplicaciones.

Un excelente tutorial para el uso de OpenCL desde el lenguaje de programación C SHARP, se tiene en Rentería (2015).

2.1.4 Algorítmica Paralela

Los algoritmos subyacen como el mecanismo base para la solución de todo problema de cualquier actividad humana, y es definida como una secuencia finita de pasos o tareas que se deben ejecutar para lograr tal propósito. Cuando se habla de algoritmos, normalmente se tiene la idea de que éstos son estrictamente secuenciales, donde las tareas se ejecutan una a continuación de la otra. Sin embargo, muchos problemas admiten una solución donde todas o algunas las tareas se pueden realizar o ejecutar al mismo tiempo, con el consecuente beneficio de la optimización de tiempo.

En el campo de las tecnologías de la información y las comunicaciones, los algoritmos paralelos dividen el problema en sub algoritmos y éstos se ejecutan en el mismo instante de

tiempo, utilizando para ello varias unidades de proceso; cada sub algoritmo resuelve una parte del problema, la suma de estas partes resuelve completamente el problema. Para los fundamentos teóricos, se tiene amplia documentación en Grama et al. (2003), Patterson y Hennessy (2014). Para la parte práctica se puede revisar la documentación de Algoritmos paralelos en Microsoft (2021).

Se debe considerar el hecho de que no todos los algoritmos son paralelizables, algunos son estrictamente secuenciales, que no admiten ningún tipo de paralelización.

Tal lo referido por Patterson y Hennessy (2014), la algorítmica paralela tomó auge en las últimas décadas, más aún recientemente con la incursión tecnológica en el campo de los procesadores multinúcleo y los GPU's. Es un área donde se han realizado y se viene realizando importantes trabajos de investigación, que han propiciado a la fecha establecer ciertos patrones de comportamiento comunes, que permiten establecer taxonomías. Se tiene mayor referencia de las diferentes características de los algoritmos paralelos en Naiouf (2004).

2.1.4.1 Taxonomía de los Algoritmos Paralelos

2.1.4.1.1 Paralelismo de Datos. Corresponde a algoritmos que efectúan procesos similares o iguales sobre estructuras de datos homogéneas, tal como los procesos numéricos efectuados sobre arreglos o matrices. El paralelismo consiste en procesar cada dato o particiones de datos de manera independiente y simultánea. Por ejemplo, se podría tener en un arreglo los RUC´s de las empresas suscritas a la Cámara de comercio, luego se desea determinar el estado de cada empresa efectuando una consulta a los servicios de la SUNAT. Este proceso se puede efectuar en paralelo, verificando cada RUC independientemente y en paralelo.

Este tipo de cómputo corresponde al modelo SIMD de la taxonomía de Flynn. Este modelo implica implementar un módulo único, el que debe ejecutarse en cada uno de los

procesadores del sistema; con la salvedad, que cada procesador opera sobre un segmento diferente de los datos.

Este tipo de paralelismo se puede implementar en una arquitectura CPU multinúcleo, así como en una arquitectura GPGPU.

2.1.4.1.2 Paralelismo de Procesos. Corresponde a algoritmos que efectúan procesos disjuntos, donde cada proceso efectúa una tarea distinta sobre datos distintos. Por ejemplo, en una aplicación de corte administrativo en el sector público, se desea enviar una cotización a diferentes proveedores, entonces un proceso se podría encargar de recuperar la lista de proveedores, evaluando en línea su situación en la SUNAT y en los organismos pertinentes que llevan el control de sanciones de las contrataciones con el estado; mientras que otro proceso se encargaría de recuperar las características y demás información relevante de los artículos, productos o servicios materia del pedido. Ambos procesos son independientes y pueden ser realizados en paralelo.

Este tipo de cómputo corresponde al modelo MIMD de la taxonomía de Flynn. Este modelo implica implementar múltiples módulos para procesar múltiples conjuntos de datos; donde, cada módulo se ejecutará de manera autónoma en cada procesador del sistema, procesando un conjunto distinto de datos.

Este tipo de paralelismo es el exponente máximo del concepto de programación paralela. Desde los inicios de la computación, las arquitecturas hardware han impulsado este tipo de paralelismo. En la infraestructura tecnológica actual que se tiene en las organizaciones, este tipo de paralelismo sólo es posible en las CPU multinúcleo, donde cada núcleo tiene autonomía para ejecutar procesos.

La arquitectura GPGPU, no permite este tipo de paralelismo.

2.1.4.2 ***Complejidad de los Algoritmos Paralelos.*** El análisis de complejidad de los algoritmos es un tema bastante estudiado. Para el análisis de complejidad de los algoritmos paralelos, se tomará como base, el análisis de complejidad general de los algoritmos; tal como se puede apreciar en la Tabla 1.

Tabla 1

Generalización de la complejidad de los algoritmos secuenciales

Orden	Descripción	Comentario
O(1)	constante	Corresponde a aquellos algoritmos cuyo tiempo de ejecución permanece constante. Por ejemplo, la suma de ***n*** números, que se efectúa con una fórmula.
O(log n)	logarítmico	A medida que crece el tamaño de los datos, el tiempo de ejecución crece logarítmicamente. Por ejemplo, la búsqueda binaria.
O(n)	Lineal	En este caso, el tiempo crece a medida que aumenta el tamaño de los datos. Por ejemplo, sumar los elementos de un arreglo unidimensional.
O(n·log(n))	n logaritmo de n	Este es un orden bueno para muchos problemas computacionales, tales como los algoritmos óptimos de ordenamiento, por ejemplo, el Quick-Sort.
$O(n^c)$	polinómico	En este orden se cumple que $c > 0$. Si $c = 2$ se dice que el orden es cuadrático, $c = 3$ el orden es cúbico, así sucesivamente. La implementación de los algoritmos que pertenecen hasta este orden, tienen sentido práctico y dentro de la teoría de la computabilidad son considerados problemas *tratables.*
$O(c^n)$	exponencial	En este orden también se cumple que $c > 0$. De aquí en adelante ya se tienen algoritmos que caen dentro de lo que la teoría de la computabilidad lo categoriza de *intratables*. El tiempo de ejecución crece muy rápidamente a medida que crece el tamaño de los datos.
O(n!)	factorial	Contempla los algoritmos de problemas aún más *intratables* que el anterior.
$O(n^n)$	combinatorio	Intratable como el anterior.

Nota. Elaboración propia.

La aplicación práctica del análisis de complejidad estriba en determinar antes de implementar a qué orden (de los mostrados en la tabla anterior) corresponde un determinado algoritmo. Se debe considerar que todos los algoritmos fundamentales, van a tener un comportamiento que se corresponda con una de las mencionadas en la tabla anterior. En

consecuencia, el análisis teórico de complejidad permitirá determinar si el tiempo de ejecución de la posible nueva propuesta es viable y será más eficiente que los existentes. Se debe enfatizar, el hecho de que los algoritmos cuyo orden corresponde al exponencial, factorial o combinatorio, son algoritmos intratables, estos crecen desmesuradamente a media que aumenta ***n***; por tanto, cuando ***n*** es grande, no existe computador que pueda ejecutar completamente el algoritmo, independiente de la rapidez del microprocesador o los recursos con los que cuente. Información más abundante de complejidad de algoritmos se tiene en los libros referentes de algorítmica, como en Cormen (2019) para la parte de complejidad de algoritmos paralelos.

Sin embargo, todo este análisis de la complejidad sólo es aplicable a los algoritmos que corresponden al modelo SISD de la taxonomía de Flynn; más propiamente a los algoritmos secuenciales que funcionan en base al modelo propuesto por Von Neumann.

Cuando se habla del análisis de complejidad de los algoritmos paralelos, se debe tener en cuenta que la paralelización de los algoritmos contribuye a disminuir los tiempos de ejecución, pero no necesariamente significa que se vaya a reducir el orden de complejidad al que pertenece el algoritmo. Es decir, si un algoritmo es de orden O(n), su implementación paralela seguirá siendo del mismo orden, sin importar el número de núcleos que se utilicen en la paralelización. En suma, si un problema (de acuerdo con la teoría de la computabilidad) está catalogado como intratable, por mucho que se paralelice a un buen número de núcleos de ejecución, seguirá siendo intratable.

En esta sección, no se pretende efectuar un análisis teórico matemático de la complejidad de algoritmos paralelos, sino, plantear algunas métricas ampliamente estudiadas para la determinación de la eficiencia de los algoritmos paralelos, tal las propuestas de Muller (2011).

La métrica básica e intuitiva viene a ser la medición del tiempo de ejecución, para lo cual se debe considerar que en una arquitectura paralela el tiempo de ejecución no solo depende

del tamaño de los datos de entrada, sino también de las características de la arquitectura, por ejemplo, del número de núcleos de procesamiento. A lo largo de la historia de la computación paralela, se han utilizado diferentes tipos de medida de la velocidad de ejecución, así, la determinación del número de Millones de instrucciones ejecutadas por segundo (MIPS) o el número de millones de instrucciones de punto flotante ejecutadas por segundo (MFlops). Existen también otros indicadores como: Dhrystone, Whestone, TPS, KLIPS, etc. El uso de este tipo de indicadores no es apropiado porque presentan una serie de detalles que hace inapropiado su uso, tal como lo manifiesta Naiouf (2004); por tanto, escapa del alcance de este trabajo la discusión de estos detalles.

Un aspecto importante que considerar son las métricas que permiten medir el rendimiento de los algoritmos paralelos. Existen herramientas que permiten analizar diferentes aspectos del rendimiento de los algoritmos paralelos; Por ejemplo, la herramienta Compute Visual Profiler que permite medir el nivel de ocupación de los procesadores, tal lo planteado por Gaudiani (2012). Sin embargo, la métrica SpeedUp es la utilizada para medir el rendimiento desde la perspectiva de tiempos de ejecución.

2.1.4.2.1 Métrica SpeedUp. La métrica SpeedUp es la más usada para determinar la eficiencia de los algoritmos paralelos, mide la ganancia de velocidad del algoritmo paralelo. De manera general se define la métrica SpeedUp S en función del tiempo de ejecución del algoritmo serial o secuencial (Ts) y el tiempo de ejecución del algoritmo paralelo (Tp).

$$S = f\,(Ts,\ Tp)$$

Y está dado por:

$$S = \frac{Ts(n)}{Tp(n)}$$

Existen varias definiciones asociadas a estos tiempos de ejecución, que derivan también en varias acepciones de la métrica *SpeedUp*, tal como se puede revisar en Sahni y Thanvantri (1996), Muller (2011) y Tosini (2015).

El *SpeedUp* absoluto considera el tiempo de ejecución del algoritmo serial más rápido *To* y el tiempo de ejecución del algoritmo paralelo ejecutado sobre N procesadores *Tp*. Está dado por:

$$SAbsoluto = \frac{To(n)}{Tp(n)}$$

Determinar el mejor algoritmo serial para la solución de un determinado problema es bastante complicado, por tanto, el cálculo del *SpeedUp* absoluto es también bastante complicado.

El *SpeedUp* algorítmico o relativo es el que más se utiliza en la práctica; considera el tiempo de ejecución del algoritmo serial T1, cuando éste es ejecutado sobre un único procesador del computador paralelo. Está dado por:

$$SRelativo = \frac{T1(n)}{Tp(n)}$$

Existen diferentes factores que limitan el *SpeedUp*, éstos deben ser considerados al momento de optimizar el algoritmo paralelo para lograr la máxima eficiencia posible. Entre los principales factores a considerar tenemos:

- El volumen de los datos a procesar por el algoritmo paralelo. Si este volumen es muy reducido o pequeño, el paralelizarlo posiblemente no ofrecerá ventajas sustanciales; podría incluso, ser menos eficiente que el algoritmo secuencial. En la práctica, los procesos paralelos requieren de un tiempo para configurar e inicializar los componentes

involucrados en el proceso paralelo; este tiempo puede ser mayor al del procesamiento en paralelo, si se efectúa sobre un conjunto de datos pequeño.

- Algoritmos o parte de ellos que no pueden paralelizarse. Hay algoritmos cuya naturaleza es eminente o parcialmente secuencial. Es relevante si la mayor parte del algoritmo es secuencial. Este caso es analizado por la ley de Amdahl.
- Procesos reiterativos efectuados sobre resultados de cálculos parciales. El riesgo en este caso es que el cálculo de estos resultados parciales al ser paralelizados se pueda efectuar varias veces en cada núcleo o procesador.
- Implementaciones ineficientes debido a que la parte paralelizable del algoritmo presente mucha complejidad lógica o dificultad en su implementación.
- El desbalance de carga, que por la naturaleza del algoritmo algunos procesadores puedan estar sobrecargados, mientras que otros estén a la espera de que los primeros terminen su trabajo.

La eficiencia calculada a partir del *SpeedUP* es otra métrica utilizada en la evaluación del rendimiento de los algoritmos paralelos; considera además el número de procesadores implicados en la ejecución del algoritmo paralelo. La eficiencia puede ser vista como el promedio de la *SpeedUp* por procesador. Está dada por:

$$E = \frac{S_p}{P}$$

El dominio de la eficiencia está dado por:

$$D(E) = [0,1]$$

Donde el valor de 1 corresponde al *SpeedUp* perfecto con un 100% de aprovechamiento.

Se puede ver también la eficiencia como el promedio del SpeedUp por procesador.

Otro aspecto importante es la Escalabilidad, que determina que un sistema es escalable si la eficiencia se mantiene en un valor fijo o constante al aumentar las variables número de procesadores y tamaño del problema.

2.1.4.2.1 Ley de Amdahl. Si el SpeedUp mide la mejora de la eficiencia del sistema, entonces, es necesario determinar matemáticamente los límites o cotas asintóticas de esta mejora. La ley de Amdahl (1967), desde una perspectiva genérica, modela matemáticamente las restricciones de mejora de un sistema en función a los componentes del sistema. Esa ley se aplica también a los sistemas paralelos, considerando que la parte algorítmica de todo sistema paralelo tiene un componente secuencial y un componente paralelo. Qué y en qué medida se restringe la mejora de los algoritmos paralelos. En el documento "Introducción y medidas de rendimiento Ley de Amdahl", de la Universidad Europea de Madrid (2017), se tiene una referencia amplia de este concepto.

Al efectuar el análisis de un algoritmo paralelo, se determina que la cota inferior del tiempo de ejecución del algoritmo está dada por la parte secuencial del algoritmo, por mucho que se maximice el número de procesadores utilizados en la parte paralela.

Si se tiene un algoritmo paralelo, donde:

P es la parte paralizable

$S = 1 - P$ es la parte secuencial

Entonces, el *SpeedUp* al trabajar con N procesadores estará limitado por:

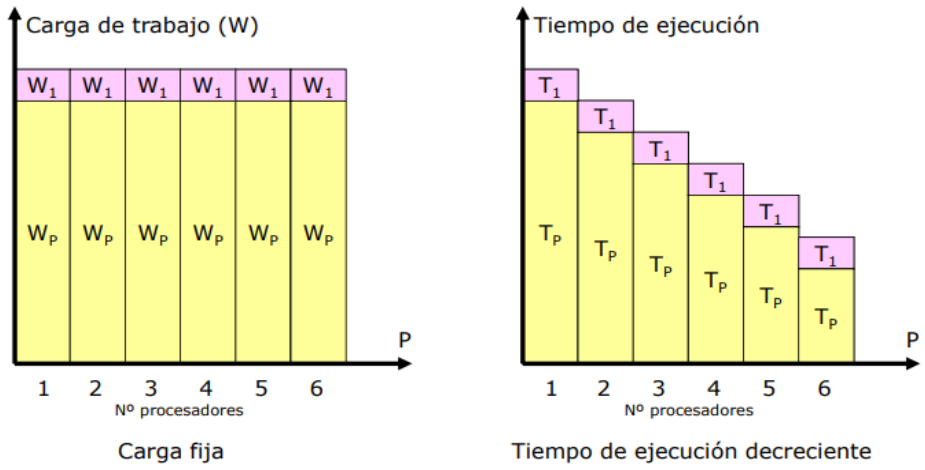
$$SN <= \frac{1}{(S + P/N)}$$

Luego, por la definición de la ley de Amdahl, el valor asintótico para *SN* es *1/S*. A partir de este valor asintótico se puede inferir que al aumentar el componente *S*, la mejora decrecerá proporcionalmente; además, para aumentar la mejora es importante reducir el componente S.

En suma, lo que plantea Admhl, es que un problema dado se podrá optimizar sólo hasta el tiempo de ejecución requerido por la parte secuencial del algoritmo. En la figura 19, se tiene una carga de trabajo secuencial dada por W1 y una carga de trabajo paralelizable dada por Wp. Como se puede apreciar, si la misma carga de trabajo se ejecuta incrementando el número de procesadores; entonces, el tiempo de ejecución disminuye, hasta alcanzar el tiempo de procesamiento T1, que corresponde a la parte secuencial del algoritmo.

Figura 19

Ley de Amdhl, limitación por carga de trabajo fija



Nota. Tosini (2015, p. 21)

Lo planteado por le ley de Amdahl tiene una perspectiva limitada a un problema; en un principio llevó a cierto escepticismo y a conjeturar que la aplicación de la computación paralela en la solución de problemas reales se circunscribiría a un conjunto reducido de problemas. Muller (2011) sin embargo, hace referencia que en la práctica se ha demostrado que el componente secuencial tiene muy poca relevancia de los problemas reales, tal como lo demostraron Gustafson (1988).

2.1.4.2.1 Ley de Gustafson-Barsis. Gustafson (1988a), efectúa una revaluación de la Ley de Amdahl, plantea una perspectiva más optimista, para la evaluación de los algoritmos que incluyen módulos paralelos.

La ley de Gustafson (1988b) no se limita al análisis de los tiempos de ejecución, considerando un único algoritmo como lo hace la Ley de Amdahl; sino, que enuncia que se puede obtener los mismos tiempos de ejecución para algoritmos que resuelven problemas de diferente tamaño; minimizando el tiempo de ejecución de la parte secuencial, a medida que crece el tamaño del problema y se incremente el número de procesadores.

La Ley de Gustafson parte de la idea que el tiempo de ejecución de un algoritmo que tiene una parte secuencial y otra paralela, está dado por:

$$T_t = T_s + T_p$$

Donde: T_s = Tiempo de ejecución de la parte secuencial

T_p = Tiempo de ejecución de la parte paralela

T_t = Tiempo total de ejecución del algoritmo

El aspecto más relevante de la Ley de Gustafson radica, en que la parte del algoritmo en paralelo puede ejecutarse en tiempo constante, si se cuenta con el número suficiente de procesadores. Entonces, si P es el número de procesadores, el tiempo total será:

$$T_t = T_s + P \,.\, T_p$$

Luego, la métrica *SpeedUp* estará dada por:

$$S(P) = (T_s + P \,.\, T_p) / (T_s + T_p)$$

Si sobre esta expresión, se considera que la fracción del tiempo empleado por la ejecución de la parte secuencial del algoritmo está dada por:

$$\alpha = Ts / (T_s + T_p)$$

Entonces, reemplazando y operando en la expresión anterior, finalmente se tiene:

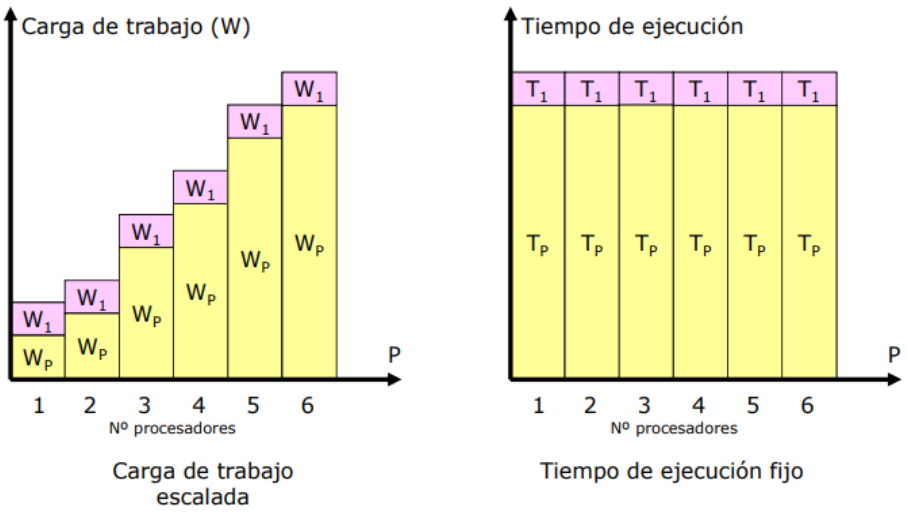
$$S(P) = P - \alpha.(P-1)$$

Que corresponde a la expresión matemática de la Ley de Gustafson; la que su puede interpretar como: Si el tiempo de ejecución de la parte secuencial del algoritmo es pequeño; entonces, se logra una aceleración muy próxima al valor de P. Este hecho se evidencia mucho más, si el tamaño del problema crece y se cuenta con el número de procesadores suficientes, la implicancia de α va a ser mínima, tendiendo el valor de S al valor de P.

En suma, lo que plantea Gustafson, es que problemas de tamaño distintos con cargas de trabajo diferentes, se pueden ejecutar en el mismo tiempo, si se cuenta con los recursos que permitan utilizar mayor número de procesadores, acorde al tamaño del problema. En la figura 20, se tiene seis problemas de diferente tamaño, con cargas de trabajo distintas; donde, cada problema tiene una carga de trabajo secuencial dada por *W1* y una carga de trabajo paralelizable dada por *Wp*. Como se puede apreciar, si a cada problema se aplica un número de procesadores en función a su tamaño; entonces, se puede alcanzar el mismo tiempo de ejecución.

Figura 20

Ley de Gustafson, limitación por tiempo fijo



Nota. Tosini (2015, p. 25)

III. MÉTODO

3.1 Tipo de Investigación

De acuerdo con el enfoque de la investigación, el presente trabajo corresponde a una investigación cuantitativa, tal como lo manifiestan Hernández y Mendoza (2018), se efectuarán mediciones y utilizarán métodos matemáticos. Así mismo, dentro de las fases del desarrollo, se identifica el problema, se analiza el marco teórico pertinente, se plantea la hipótesis que será sometida a contrastación, que mediante el procesamiento de datos permitirá obtener e interpretar resultados, a partir de los cuáles se determinará las conclusiones.

De acuerdo con la finalidad de la investigación, el trabajo es catalogado como de tipo de investigación aplicada; porque, se hace uso de las teorías y tecnologías vigentes para resolver un problema práctico, que contribuya a mejorar un determinado contexto. Específicamente, se trata de aplicar y aprovechar los avances y las ventajas de las arquitecturas heterogéneas, en soluciones informáticas determinísticas de corte administrativo o de gestión, presentes en toda organización. Cabe resaltar, que hoy en día estas tecnologías sólo se circunscriben a resolver problemas científicos o tecnológicos en contextos eminentemente estocásticos.

De acuerdo con el alcance de la investigación, ésta es de tipo correlacional, porque se relacionan tres variables: El tipo de arquitectura heterogénea, el número de procesadores y el volumen de información a procesar; analizándose para cada caso las ventajas y desventajas.

De acuerdo con el diseño de la investigación, ésta corresponde a una investigación experimental, porque se variará el tipo de arquitectura heterogénea y el volumen de información a procesar; luego, se verá el impacto de esta variación en las diferentes arquitecturas heterogéneas; finalmente, se aplicará regresión para determinar el

comportamiento de los tiempos de procesamiento para diferentes volúmenes de datos y en los diferentes tipos de arquitecturas.

3.2 Población y Muestra

Población

Las organizaciones cuentan con ingentes volúmenes de datos de gestión, sobre los que se puede aplicar procesos de deducción o inferencia; a partir de los cuales se puede explicitar información implícita, que coadyuvaría a la mejor toma de decisiones.

El trabajo pretende mostrar las ventajas de las arquitecturas heterogéneas en los procesos de deducción de los datos de gestión de las organizaciones; en consecuencia, es factible utilizar la totalidad de los datos, dado que éstos serán procesados por un computador como parte del proceso de experimentación.

Por tanto, la población estará constituida por la totalidad de los registros de la base de datos de la organización, sobre las que serán aplicados los procesos de deducción de información implícita de las bases de datos.

Para efectos de medir los tiempos de procesamiento en las diferentes arquitecturas heterogéneas en el prototipo implementado, se efectuarán procesos sobre diferentes volúmenes de datos; de modo que, en cada ejecución la población variará desde los 10,000 hasta los 6’000,000 de registros.

Muestra

Una empresa dedicada al desarrollo de software puede implementar diferentes procesos de inferencia de datos. En el caso del presente proyecto, se consideran procesos de inferencia referidos a ventas; por tanto, la muestra está constituida por la totalidad de los datos de ventas que serán utilizados en el proceso de deducción de información implícita de las bases de datos para identificar el tipo de venta al que pertenece cada venta.

3.3 Operacionalización de Variables

El detalle de la operacionalización de variables se muestra en la tabla 2.

Tabla 2

Operacionalización de variables

VARIABLES	DIMENSIONES	INDICADORES	ESCALA	MEDIDA
VARIABLE INDEPENDIENTE				
Motor de inferencia paralelo en una arquitectura heterogénea (Multinúcleo y GPGPU)	1. Tipo arquitectura	1.1. Procesador de tipo multinúcleo o de tipo GPU	Nominal	Multinúcleo/GP GPU
		1.2. Número de núcleos	Razón	Número de unidades
Volumen de datos	2. Tuplas de la base de hechos	2.1. Número de registros	Razón	Número de unidades
VARIABLE DEPENDIENTE				
Optimización en la deducción de información implícita en las bases de datos	3. Características de un motor de inferencia paralelo en arquitectura heterogénea	3.1. Características de un motor de inferencia paralelo en arquitectura multinúcleo	Nominal	Características
		3.2. Características de un motor de inferencia paralelo en arquitectura GPGPU	Nominal	Características
	4. Tiempo de optimización		Razón	Milisegundos

	4.1. Disminución del tiempo de procesamiento en arquitectura multinúcleo	Razón	Milisegundos
	4.2. Disminución del tiempo de procesamiento en arquitectura GPGPU		
5. Eficiencia de optimización	5.1 Número de veces más rápido de procesamiento en una arquitectura multinúcleo respecto al procesamiento secuencial.	Razón	Número de veces
	5.2 Número de veces más rápido de procesamiento en una arquitectura GPGPU respecto al procesamiento secuencial.	Razón	Porcentaje
6. Proyección de los tiempos de procesamiento para cualquier volumen de datos	6.1 Proyección de tiempos de procesamiento para cualquier volumen de datos en una arquitectura multinúcleo.	Razón	Milisegundos
	6.2 Proyección de tiempos de procesamiento para cualquier volumen de datos en una arquitectura GPGPU.	Razón	Milisegundos

3.4 Instrumentos

Como instrumento para la recolección de datos y posterior análisis se utiliza una ficha de observación estructurada, donde se consigna los tiempos de ejecución, así como la eficiencia de los procesos de optimización tanto de la arquitectura multinúcleo, así como de la GPGPU. La referida ficha está en formato Excel.

3.5 Procedimientos

Se realizará primero la **Investigación documental**, revisando los antecedentes relacionados al tema; así como, los conceptos teóricos que sustenten el trabajo, conceptos como: computación paralela, motores de inferencia y arquitectura heterogénea.

Luego, se procederá con la fase de la **investigación experimental**; para lo cual, Se idea, diseña e implementa una propuesta de un modelo de un motor de inferencia paralelo en una arquitectura heterogénea compuesta por CPU multinúcleo y GPGPU; utilizando como base sentencias lógicas expresadas en forma normal conjuntiva (FNC), soportadas en estructuras de datos unidimensionales; dado que, la arquitectura GPGPU no soporta algoritmos recursivos y estructuras de datos multidimensionales. La validación se realiza mediante la implementación de un prototipo construido en base al motor de inferencia paralelo; con dicho prototipo, se experimenta ejecutando procesos de inferencia sobre diferentes volúmenes de datos de una base de datos transaccional. Los tiempos de optimización obtenidos, así como las eficiencias logradas, se consignan en sus respectivas tablas.

Finalmente, se efectuará un proceso de regresión y correlación para determinar funciones matemáticas que permitan proyectar tiempos de procesamiento en función al número de tuplas, sin tener que efectuar el proceso experimental.

3.6 Análisis de Datos

Los datos plasmados en las fichas de observación serán presentados en forma de tablas y gráficos; luego, serán analizadas aplicando la métrica ***SpeedUp*** para determinar la eficiencia del procesamiento del motor de inferencia paralelo; también, se utilizará herramientas software como las funciones estadísticas de Excel.

IV. RESULTADOS

En este capítulo se muestran los resultados obtenidos en el presente trabajo. Se considera un aspecto relevante, la propuesta del motor de inferencia paralelo en arquitectura heterogénea, enfatizándose en las características que debe tener éste, considerando sobre todo su aplicabilidad a la arquitectura GPGPU. Luego del diseño e implementación del motor de inferencia paralelo, así como de la implementación del prototipo, en la fase de ejecución, se considera prioritariamente el tipo de arquitectura como la variable independiente de mayor relevancia. Se efectúan los procesos de inferencia, considerando el procesador del tipo CPU, que se toma como base para las respectivas comparaciones de disminución de tiempos y optimización de los procesos de inferencia; luego, se consideran los procesadores de tipo multinúcleo y GPGPU para determinar en qué medida se disminuyen los tiempos de procesamiento y se logra la optimización.

Otra variable independiente determinante es el volumen de datos sobre los que se efectúa el proceso de inferencia, considerando que mayores volúmenes de datos implican mayores tiempos de procesamiento. Así mismo, la expectativa de requerir procesos de optimización aplicando computación paralela, se pone de manifiesto en contextos donde se tiene grandes volúmenes de datos. En esta intención y para lograr conclusiones más objetivas, se considera volúmenes de datos que van desde los cientos de miles de tuplas hasta los millones.

4.1 Propuesta de un Modelo de un Motor de Inferencia Paralelo

La evolución natural de los motores de inferencia, en un inicio se orientó hacia la inteligencia artificial, más específicamente a la implementación de sistemas expertos. La principal característica de este tipo de problemas es la complejidad de los algoritmos requeridos; estos motores de inferencia requieren reglas con varios niveles de anidamiento y la

recursión como parte fundamental de los procesos de deducción. Desde la perspectiva de diseño e implementación de estos motores, se usa el modelo SISD de la taxonomía de Flynn; que se implementa suficientemente en una arquitectura de un solo CPU.

Con el incremento de las expectativas y las necesidades de la sociedad en las tecnologías de la información, cada vez se requiere procesar los grandes volúmenes de datos subyacentes en las organizaciones. Los responsables de tomar decisiones desde los niveles operativos hasta los niveles gerenciales consideran que la información es también un recurso estratégico; por tanto, requieren información no sólo a nivel de reportes o resúmenes estadísticos; sino, también información en línea que pueda ser inferida de la masa de datos con la que cuentan. Una característica relevante en este ámbito es que se requiere incluir en el software operativo de la organización, funcionalidades que satisfagan en tiempo real estas necesidades de información. Es en este contexto, que se advierte la necesidad de motores de inferencia veloces, con una gran capacidad de cómputo; que prioritariamente efectúen procesos de deducción sobre un gran volumen de datos. También, es en este contexto, que se percibe como una respuesta tecnológica adecuada, el uso de la computación heterogénea ya sea CPU multinúcleo o GPGPU; que posibilita la implementación de motores de inferencia paralelos.

La tendencia posiblemente sea, el uso de motores de inferencia paralelos basados en los modelos SIMD o SIMT, para problemas cuya lógica de deducción sea simple, aplicado a grandes volúmenes de datos; quedando siempre latente el uso de motores de inferencia basados en el modelo SISD, para problemas cuya lógica de deducción sea compleja.

4.1.1 Análisis de las Características del Modelo del Motor de Inferencia Paralelo

Como se mencionó, las actuales computadoras cuentan con una gran capacidad de cómputo y de naturaleza heterogénea, con diferentes tipos de unidades de procesamiento, tal como el CPU principal, los núcleos CPU y las GPGPU. El modelo del motor de inferencia paralelo debe ser capaz de potenciar el uso de estas diferentes unidades de procesamiento. Por

tanto, se debe realizar un análisis de las ventajas y desventajas de las diferentes arquitecturas que soportan procesamiento paralelo, tal como se muestra en la tabla 3.

Tabla 3

Comparativa de ventajas y desventajas de arquitecturas Heterogéneas

COMPARATIVA DE ARQUITECTURAS HETEROGÉNEAS		
	Núcleos CPU	**GPGPU**
Ventajas	- Soporta modelo MIMD (Paralelismo de procesos y de datos) - Permite recursión - Procesa cualquier tipo de datos - Utiliza cualquier tipo de estructura de datos	- Cuenta con cientos o miles de unidades de procesamiento - Alta capacidad de cómputo - Alta velocidad de procesamiento
Desventajas	- Tiene poco más o menos de una decena de unidades de procesamiento - Capacidad de cómputo menor - Velocidad de procesamiento menor	- Solo soporta modelo SIMT (Paralelismo de datos) - No permite recursión - Procesa solo datos numéricos - Utiliza solo estructuras de datos lineales

Nota. Elaboración propia.

Dado que el presente trabajo está orientado a coadyuvar a mejorar las potencialidades de las aplicaciones de gestión administrativa, entonces se debe analizar las características de este tipo de aplicaciones; de modo que, en base a estas características determinar las especificaciones del motor de inferencia paralelo. Este análisis se ilustra en la tabla 4.

Tabla 4

Análisis de las características del software de gestión administrativa

ANÁLISIS DEL SOFTWARE DE GESTIÓN ADMINISTRATIVA	
Características	**Especificaciones técnicas**
Cada vez se cuenta con grandes volúmenes de datos en bases de datos transaccionales	Alta capacidad de cómputo
Consultas sobre el mismo conjunto de datos	Paralelismo de datos
Procesos en línea, con tiempos de respuesta cortos	Alta velocidad de procesamiento
Diferentes tipos de datos	Datos categóricos transformados a numéricos
Consultas con sentencias lógicas anidadas	Sentencias lógicas en forma normal conjuntiva

Nota. Elaboración propia.

Del análisis efectuado en la tabla 4 se determina que lo más pertinente para plantear el modelo del motor de inferencia paralelo es la arquitectura GPGPU. Lo que implicaría que las organizaciones que deseen utilizar software de gestión administrativa potenciados con motores de inferencia paralela requerirían además de la GPU propia de la máquina contar en lo posible con tarjetas del tipo dGPU (Unidad de procesamiento gráfico dedicada). Sin embargo, las organizaciones no todas cuentan con grandes volúmenes de datos; por ejemplo, algunas microempresas con bajos volúmenes de datos, podrían trabajar sólo con la CPU principal o con los Núcleos CPU.

En consecuencia, se plantea un modelo que pueda realizar los procesos de inferencia en las diferentes unidades de procesamiento con la que cuenta el servidor; esto es, plantear motores de inferencia sobre el CPU principal, sobre los núcleos CPU y sobre el GPGPU; de modo que se pueda procesar convenientemente diferentes volúmenes de datos; además, los desarrolladores puedan optar por el motor de inferencia más apropiado para la aplicación que estén desarrollando.

4.1.2 Diseño del Motor de Inferencia Paralelo

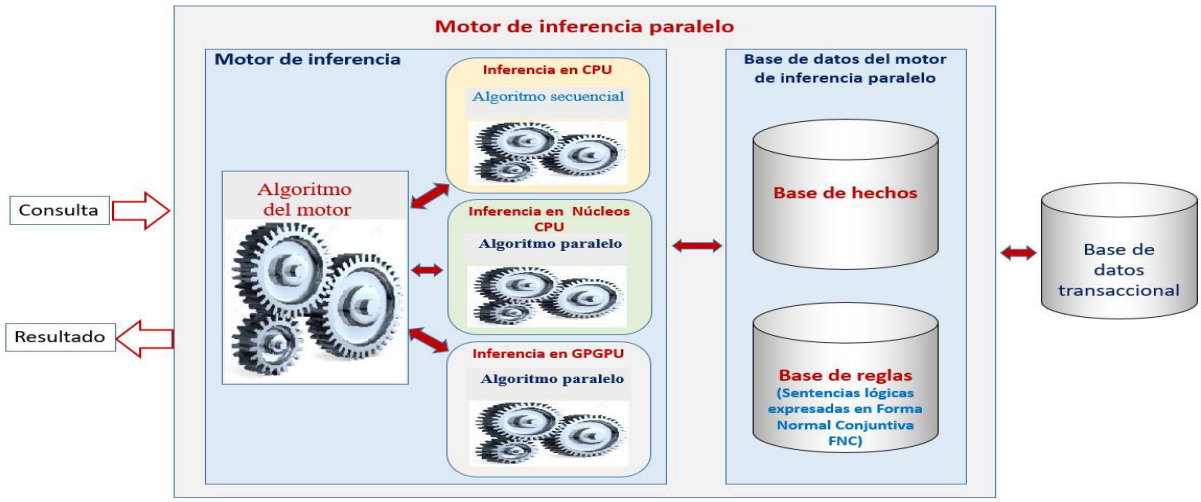
En las arquitecturas heterogéneas, específicamente en la arquitectura que hace uso del gran potencial de cómputo de las GPGPU, no es posible hacer uso de la recursión; en consecuencia, los procesos de deducción no pueden ser anidados y se ejecutan en un solo nivel de inferencia de reglas. Esta restricción requiere el uso de métodos de inferencia como el de "Resolución" que hace uso de las formas normales clausuladas como la forma normal conjuntiva (FNC). Por tanto, el uso de este tipo de motores se orienta a procesos de inferencia de grandes volúmenes de datos con algoritmos de complejidad simple. Por otra parte, el proceso de inferencia se potencia, con la gran capacidad de cómputo que ofrecen los cientos o miles de unidades de procesamiento de estas GPGPUs, que posibilitará deducciones que

satisfagan requerimientos en tiempo real; sin dejar de lado las otras arquitecturas, que pueden funcionar también con las mismas restricciones y especificaciones de las GPGPUs.

4.1.2.1 Modelo Propuesto del Motor de Inferencia Paralelo. En base a las características y especificaciones descritas en la sección anterior se propone el modelo de la figura 21.

Figura 21

Modelo de un motor de inferencia paralelo



Nota. Elaboración propia.

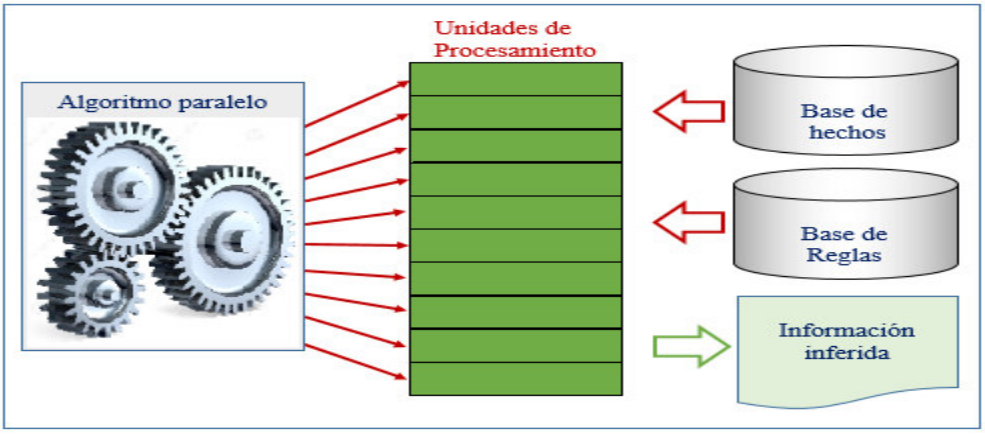
Como las organizaciones consideran los datos como un recurso estratégico, entonces, la tendencia es que cada vez se tengan mayores volúmenes de datos, con una tendencia creciente. En consecuencia, se debe propender prioritariamente al uso de la arquitectura GPGPU. Por tanto, el diseño y la implementación del motor de inferencia paralelo, estará condicionado a satisfacer los requerimientos técnicos de esta arquitectura, considerando específicamente el modelo de cómputo del tipo SIMD o SIMT.

El modelo propuesto se sugiere la implementación como una *"Interfaz de Programación de Aplicaciones - API"*, de modo que pueda ser utilizada por cualquier aplicación.

4.1.2.2 Componentes de un Motor de Inferencia Paralelo. El componente principal de un motor de inferencia paralelo son los procesadores que pueden ser del tipo CPU multinúcleo o GPGPU. El modelo de cómputo es del tipo SIMD o SIMT. Esto implica, que se dispone de un conjunto único de instrucciones implementados como un módulo; éste es asignado a cada procesador, donde cada procesador ejecutará el mismo conjunto de instrucciones, trabajando sobre un segmento diferente de la base de hechos, aplicando sobre éstos las reglas de la base de reglas. La figura 22 ilustra este hecho.

Figura 22

Componentes de un motor de inferencia paralelo



Nota. Elaboración propia.

4.1.2.2.1 Unidades de Procesamiento. Para efectos del presente trabajo, sólo se consideran las arquitecturas de CPU Multinúcleo y GPGPU. No se considera las arquitecturas multi CPU, ni FPGA, ni ASIC; dado que éstas al ser caras, no son de uso frecuente en las organizaciones comerciales, industriales o de servicios.

La funcionalidad de las unidades de procesamiento de ambas arquitecturas, requieren de un módulo único que se asigna a cada procesador; éste se ejecuta de manera paralela y autónoma, con la particularidad que cada uno procesa un segmento específico de datos de la base de hechos.

4.1.2.2.2 Algoritmo Paralelo. Un motor de inferencia tiene varias unidades lógicas o módulos, la mayoría de ellos de ejecución secuencial, realizado por el CPU principal. Por ejemplo, la inicialización del motor de inferencia, la interacción con la base de datos deductiva, la declaración de las estructuras de datos requeridas, etc. Sin embargo, si se desea mejorar los tiempos de procesamiento y satisfacer necesidades de información en tiempo real, se debe identificar la parte del algoritmo que se tiene que ejecutar de manera paralela en cada una de las unidades de procesamiento. Esta parte del algoritmo es que se implementará de modo paralelo y constituirá la base para el motor de inferencia paralelo.

El diseño e implementación del algoritmo paralelo, dependerá del tipo de arquitectura utilizado, así como del tipo de tarjeta GPU usado; que determinará también las herramientas y el lenguaje de programación utilizado.

4.1.2.2.3 Base de Datos de Hechos. Las organizaciones cuentan con la información operativa interna y externa, que en algunos casos se pueden constituir en grandes volúmenes de información. Es de esperar, que se pueden obtener ventajas competitivas de la explotación de esta información. Entonces, la organización, debe avizorar y pergeñar el tipo de información a obtener de los procesos de deducción, en función de la cuál debe estructurar una base de hechos.

La base de hechos debe ser estructurada para lograr objetivos específicos, como se ilustrará más adelante en la implementación del prototipo.

4.1.2.2.4 Base de Datos de Reglas. Los procesos de inferencia o deducción obedecen a una serie de condicionantes, que desde la perspectiva informática se expresan como sentencias lógicas. En una base de conocimientos, estas sentencias se formalizan como reglas. Entonces, la base de reglas almacena las sentencias lógicas (reglas) en base a las cuales se efectuará el proceso de inferencia.

Las reglas son específicas para cada problema; por tanto, son específicas para cada proceso de deducción.

4.1.2.3 ***Especificación Técnica de las Interfaces de Programación de Aplicaciones (API).*** En esta sección se desarrolla la especificación técnica de los componentes software del motor de inferencia paralelo, subsumidos dentro de una base de datos deductiva. Para ello, se enmarca esta especificación en el paradigma orientado a objetos.

Existen diferentes lenguajes en el modelado de componentes software, tal como el popular Lenguaje Unificado de modelamiento (UML – Unified Modeling Language) patrocinado por la OMG (2017). Sin embargo, este lenguaje está más orientado al desarrollo de software de usuario final. Es también propósito del presente trabajo, una propuesta de un motor de inferencia paralelo que con ligeras adecuaciones pueda ser utilizado en diferentes contextos; para lo cual, los desarrolladores de software de gestión deberán subsumirlo como parte de los módulos de la aplicación software. Es decir, se intenta proponer el diseño e implementación de interfaces de programación de aplicaciones (Application Programming Interfaces – API) con la funcionalidad de un motor de inferencia paralelo. En consecuencia, se utilizará una técnica más pertinente a las especificaciones de clases para la implementación de API's.

En este sentido se hará uso de la técnica propuesta por Beck y Cunningham (1989), denominada tarjetas CRC (Clase – Responsabilidad – Colaboración); acrónimo que enfatiza el hecho de que las ***clases*** son abstracciones que modelan objetos tangibles o intangibles del mundo real; además, éstas tienen ***responsabilidades*** y que actúan en ***colaboración*** con objetos de otras clases para dar solución a un problema. La figura 23 ilustra la estructura de una tarjeta CRC.

Figura 23

Estructura de una tarjeta CRC

NOMBRE DE LA CLASE	
RESPONSABILIDADES (Describe los servicios que ofrece la clase para interactuar con otras clases)	**COLABORACIÓN** (Describe las otras clases con las que interactúa esta clase)

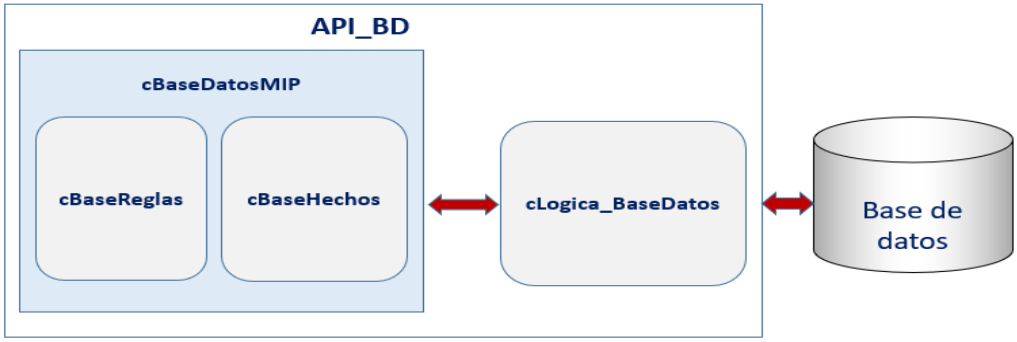
Para alcanzar los propósitos del presente trabajo, se diseñan e implementan dos API's: La primera para implementar las funcionalidades de interacción de la base de datos deductiva; la segunda, para implementar las funcionalidades del motor de inferencia paralelo. Estas API's se implementan como librerías de enlace dinámico (Dynamic Link Library - DLL). Los identificadores asignados a cada API son los siguientes:

- API_BD (Base de datos del motor de inferencia paralelo)
- API_MIP

4.1.2.3.1 Especificación Técnica de la API_BD. Esta API es la responsable de interactuar con la base de datos del motor de inferencia paralelo, recuperando los datos de la base de hechos y de la base de reglas, para luego asignarlas al motor de inferencia paralelo. El esquema gráfico de las clases que componen esta API se muestra en la figura 24:

Figura 24

Diagrama de bloques de la API_BD



Nota. Elaboración propia.

A continuación, se documentan las clases de esta API, utilizando las tarjetas CRC. Las responsabilidades se especificarán en términos de atributos y métodos. Mientras que, en el área de colaboración, se listará las clases con las que interactúa la clase.

A) Clase cBaseDatosMPI

Las especificaciones técnicas de esta clase, se muestran en la figura 25.

Figura 25

Especificación de la clase cBaseDatosMPI en tarjeta CRC

Clase: cBaseDatosMIP	
Responsabilidades	**Colaboración**
Atributos: aBaseHechos aBaseReglas aMotorInferenciaParalelo **Métodos:** Inicializar () ProcesarInferenciaCPU() ProcesarInferenciaNUCLEOS() ProcesarInferenciaGPU() ProcesarInferenciaEnArquitecturaHeterogenea()	 cBaseHechos cBaseReglas cMotorInferenciaParalelo

Responsabilidades

Esta clase abstrae la parte lógica o de proceso de la base de datos del motor de inferencia paralelo; para lo cual, gestiona objetos correspondientes a los componentes de la base de datos deductiva, como son: base de hechos, base de reglas y el motor de inferencia. Requiere que en el constructor se le suministre la conexión de base de datos física, de donde debe obtener los hechos y las reglas con las que efectuará los diferentes procesos de inferencia.

Atributos:

aBaseHechos de tipo cBaseHechos. Viene a ser el objeto que subsume y gestiona en memoria principal los hechos de la base de datos.

aBaseReglas de tipo cBaseReglas. Viene a ser el objeto que subsume y gestiona en memoria principal las reglas de la base de datos.

aMotorInferenciaParalelo de tipo cMotorInferenciaParalelo. Viene a ser el objeto que interactúa con la lógica del proceso de inferencia, aplicando las reglas sobre los hechos.

Métodos:

Inicializar

Parámetros de entrada: No tiene

Parámetros de salida: No tiene.

Descripción

Inicializa los componentes de la base de datos deductiva. Activa la base de datos de hechos y de reglas; luego, mediante éstas, recupera los hechos y las reglas de la base de datos física.

Inicializa también el motor de inferencia paralelo, que subsume los diferentes tipos de inferencia, que se puede realizar en una arquitectura heterogénea.

ProcesarInferenciaCPU

Parámetros de entrada: No tiene

Parámetros de salida: Tiempo de ejecución de tipo entero largo.

Descripción

A través de un objeto de la clase cMotorInferenciaParalelo, invoca al proceso de inferencia que se ejecuta en la arquitectura CPU del equipo. Devuelve el tiempo empleado en la ejecución del proceso de inferencia.

ProcesarInferenciaNUCLEOS

Parámetros de entrada: No tiene

Parámetros de salida: Tiempo de ejecución de tipo entero largo.

Descripción

A través de un objeto de la clase cMotorInferenciaParalelo, invoca al proceso de inferencia que se ejecuta en la arquitectura de los Núcleos CPU existentes en el equipo. Devuelve el tiempo empleado en la ejecución del proceso de inferencia.

ProcesarInferenciaGPU

Parámetros de entrada: No tiene

Parámetros de salida: Tiempo de ejecución de tipo entero largo.

Descripción

A través de un objeto de la clase cMotorInferenciaParalelo, invoca al proceso de inferencia que se ejecuta en la arquitectura GPU del equipo. Devuelve el tiempo empleado en la ejecución del proceso de inferencia.

ProcesarInferenciaEnArquitecturaHeterogenea

Parámetros de entrada: No tiene

Parámetros de salida: Tiempo_CPU, Tiempo_NUCLEOS y Tiempo_GPI, de tipo entero largo.

Descripción

A través de un objeto de la clase cMotorInferenciaParalelo, invoca a los procesos de inferencia que se ejecuta en tres de las arquitecturas (CPU, NÚCLEOS y GPU) existentes en los equipos de cómputo utilizados hoy en día en las organizaciones. Devuelve los tiempos empleados en la ejecución de los procesos de inferencia en cada arquitectura.

Colaboración

Con ***cBaseHechos***. Un objeto de la clase cBaseHechos recupera los hechos de la base de datos física, mediante un objeto de la clase cLogica_BaseDatos.

Con ***cBaseReglas***. Un objeto de la clase cBaseReglas recupera los hechos de la base de datos física, mediante un objeto de la clase cLogica_BaseDatos.

Con ***cMotorInferenciaParalelo***. La base de datos deductiva interactúa con un objeto de esta clase para efectuar los diferentes procesos de inferencia.

Con ***cLogica_BaseDatos***. Un objeto de esta clase es requerido por los objetos de hechos y reglas que deben recuperar sus datos desde la base de datos física.

B) Clase cBaseHechos

Las especificaciones técnicas de esta clase, se muestran en la figura 26.

Figura 26

Especificación de la clase cBaseHechos en tarjeta CRC

Clase: cBaseHechos	
Responsabilidades	**Colaboración**
Atributos: aNroFilas aNroColumnas aHechos aLogicaBD **Métodos:** DatosBaseDeHechos() RecuperarHechos()	cBaseDatosMIP

Responsabilidades

Esta clase abstrae la parte correspondiente a la gestión de la base de hechos en memoria, implementado en estructuras de datos de tipo arreglo bidimensional, para ser procesados por las arquitecturas heterogéneas existentes en el equipo de cómputo.

Atributos:

aNroFilas de tipo entero. Determina el número de filas que tiene la estructura de datos bidimensional que almacena los hechos.

aNroColumnas de tipo entero. Determina el número de columnas que tiene la estructura de datos bidimensional que almacena los hechos.

aHechos de tipo arreglo bidimensional de reales. Estructura de datos bidimensional para almacenar los hechos de la base de datos.

Métodos:

DatosBaseDeHechos

Parámetros de entrada: No tiene.

Parámetros de salida: Tabla de datos.

Descripción

Recupera los hechos de la base de datos física y lo retorna en una tabla; sin efectuar ninguna transformación, normalización o discretización.

RecuperarHechos

Parámetros de entrada: No tiene.

Parámetros de salida: No tiene.

Descripción

Recupera los hechos de la base de datos física efectuando procesos de transformación como normalización a números reales y discretización, de modo que puedan ser utilizados sobre todo en procesadores de tipo GPGPU. Los datos se almacenan en el atributo aHechos.

Colaboración

Con ***cBaseDatosMIP***. Un objeto de esta clase es la que articula los objetos tanto de la clase cBaseHechos y cBaseReglas.

Con ***cLogica_BaseDatos***. Un objeto de esta clase es utilizado por la base de hechos para recuperar los hechos de la base de datos física.

C) Clase cBaseReglas

Las especificaciones técnicas de esta clase, se muestran en la figura 27.

Figura 27

Especificación de la clase cBaseReglas en tarjeta CRC

Clase: cBaseReglas	
Responsabilidades	**Colaboración**
Atributos: aNroReglas aNroMaxCondicionesRegla aReglas aLogicaBD **Métodos:** DatosBaseDeReglas() RecuperarReglas()	cBaseDatosMIP

Responsabilidades

Esta clase abstrae la parte correspondiente a la gestión de la base de reglas en memoria, implementado en una estructura de datos de tipo arreglo bidimensional, para ser procesados por las arquitecturas heterogéneas existentes en el equipo de cómputo.

Atributos:

aNroReglas de tipo entero. Determina el número de reglas que tiene la estructura de datos bidimensional correspondiente a las reglas.

aNroMaxCondicionesRegla de tipo entero. Cada regla está implementada con un determinado número de condiciones. En la estructura de datos se almacena la regla y las condiciones asociadas a cada regla; en consecuencia, es necesario saber el número de condiciones que tiene la regla con el mayor número de condiciones.

aReglas de tipo arreglo bidimensional de reales. Estructura de datos bidimensional para almacenar las reglas de la base de datos.

Métodos:

DatosBaseDeReglas

Parámetros de entrada: No tiene.

Parámetros de salida: Tabla de datos.

Descripción

Recupera las reglas de la base de datos física y lo retorna en una tabla; sin efectuar ninguna transformación ni adecuación.

RecuperarReglas

Parámetros de entrada: No tiene.

Parámetros de salida: No tiene.

Descripción

Recupera las reglas de la base de datos física efectuando procesos de transformación como normalización a números reales y discretización de los operadores lógicos a números enteros, de modo que puedan ser utilizados sobre todo en procesadores de tipo GPGPU. Los datos se almacenan en el atributo aReglas.

Colaboración

Con ***cBaseDatosMIP***. Un objeto de esta clase es la que articula los objetos tanto de la clase cBaseHechos y cBaseReglas.

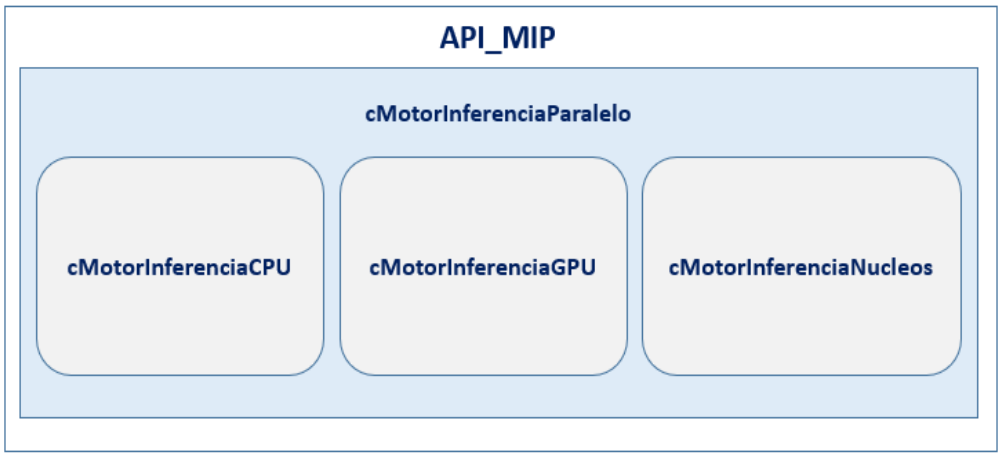
Con ***cLogica_BaseDatos***. Un objeto de esta clase es utilizado por la base de reglas para recuperar las reglas de la base de datos física.

4.1.2.3.2 Especificación Técnica de la API_MIP

Esta API es la responsable de efectuar el proceso de inferencia en las diferentes arquitecturas existentes en el servidor de la base de datos deductiva. El gráfico de las clases que componen esta API se muestra en la figura 28.

Figura 28

Diagrama de bloques de la API MotorInferenciaParalelo



Nota. Elaboración propia.

A continuación, se documentan las clases de esta API, utilizando las tarjetas CRC. Las responsabilidades se especificarán en términos de atributos y métodos. Mientras que, en el área de colaboración, se listará las clases con las que interactúa la clase.

A) Clase cMotorInferenciaCPU

Las especificaciones técnicas de esta clase, se muestran en la figura 29.

Figura 29

Especificación de la clase cMotorInferenciaCPU en tarjeta CRC

Clase: cMotorInferenciaCPU	
Responsabilidades	**Colaboración**
Atributos: aHechos aReglas aNroReglas **Métodos:** ProcesarInferenciaCPU	cMotorInferenciaParalelo

Responsabilidades

Esta clase es la responsable de procesar las reglas y aplicarlas sobre los hechos, para efectuar el proceso de inferencia en una arquitectura convencional CPU.

Atributos:

aHechos de tipo arreglo bidimensional de números reales. Representa a la base de hechos, con los datos normalizados a valores numéricos reales y discretizados para su procesamiento.

aReglas de tipo arreglo bidimensional de números reales. Representa a la base de reglas, con las condiciones de cada regla, normalizados a una estructura bidimensional, donde en cada fila se representa una regla con todas sus condiciones.

Métodos:

ProcesarInferenciaCPU

Parámetros de entrada:	Hechos de tipo arreglo bidimensional.
	Reglas de tipo arreglo bidimensional.
	NroReglas de tipo entero.
Parámetros de salida:	Resultado de tipo arreglo unidimensional de enteros.
	Tiempo de tipo entero largo.

Descripción

Procesa las reglas y aplica las condiciones de éstas a la base de hechos, para efectuar el proceso de inferencia; para lo cual, utiliza un cronómetro que permite medir el tiempo de procesamiento, devolviendo este resultado en milisegundos.

Colaboración

Con ***cMotorInferenciaParaleo***. Dentro del contexto de inferencia en arquitectura heterogénea, es desde un objeto de esta clase que se ejecuta el objeto de la clase

cMotorInferenciaCPU, para comparar con los procesos de inferencia de las otras arquitecturas.

B) Clase cMotorInferenciaNUCLEOS

Las especificaciones técnicas de esta clase, se muestran en la figura 30.

Figura 30

Especificación de la clase cMotorInferenciaNUCLEOS en tarjeta CRC

Clase: cMotorInferenciaNUCLEOS	
Responsabilidades	**Colaboración**
Atributos: aHechos aReglas aNroReglas **Métodos:** ProcesarInferenciaNUCLEOS	cMotorInferenciaParalelo

Responsabilidades

Esta clase es la responsable de procesar las reglas y aplicarlas sobre los hechos, para efectuar el proceso de inferencia en una arquitectura NUCLEOS-CPU.

Atributos:

aHechos de tipo arreglo bidimensional de números reales. Representa a la base de hechos, con los datos normalizados a valores numéricos reales y discretizados para su procesamiento.

aReglas de tipo arreglo bidimensional de números reales. Representa a la base de reglas, con las condiciones de cada regla, normalizados a una estructura bidimensional, donde en cada fila se representa una regla con todas sus condiciones.

Métodos:

ProcesarInferenciaNUCLEOS

Parámetros de entrada: Hechos de tipo arreglo bidimensional.

Reglas de tipo arreglo bidimensional.

NroReglas de tipo entero.

Parámetros de salida: Resultado de tipo arreglo unidimensional de enteros.

Tiempo de tipo entero largo.

Descripción

Procesa las reglas y aplica las condiciones de éstas a la base de hechos, para efectuar el proceso de inferencia; para lo cual, utiliza un cronómetro que permite medir el tiempo de procesamiento, devolviendo este resultado en milisegundos.

Colaboración

Con ***cMotorInferenciaParaleo***. Dentro del contexto de inferencia en arquitectura heterogénea, es desde un objeto de esta clase que se ejecuta el objeto de la clase cMotorInferenciaNUCLEOS, para comparar con los procesos de inferencia de las otras arquitecturas.

C) Clase cMotorInferenciaGPU

Las especificaciones técnicas de esta clase, se muestran en la figura 31.

Figura 31

Especificación de la clase cMotorInferenciaGPU en tarjeta CRC

Clase: cMotorInferenciaGPU	
Responsabilidades	**Colaboración**
Atributos: aHechos aReglas aNroReglas **Métodos:** ProcesarInferenciaGPU	cMotorInferenciaParalelo

Responsabilidades

Esta clase es la responsable de procesar las reglas y aplicarlas sobre los hechos, para efectuar el proceso de inferencia en una arquitectura GPU.

Atributos:

aHechos de tipo arreglo bidimensional de números reales. Representa a la base de hechos, con los datos normalizados a valores numéricos reales y discretizados para su procesamiento.

aReglas de tipo arreglo bidimensional de números reales. Representa a la base de reglas, con las condiciones de cada regla, normalizados a una estructura bidimensional, donde en cada fila se representa una regla con todas sus condiciones.

Métodos:

ProcesarInferenciaGPU

Parámetros de entrada: Hechos de tipo arreglo bidimensional.
Reglas de tipo arreglo bidimensional.
NroReglas de tipo entero.

Parámetros de salida: Resultado de tipo arreglo unidimensional de enteros.
Tiempo de tipo entero largo.

Descripción

Procesa las reglas y aplica las condiciones de éstas a la base de hechos, para efectuar el proceso de inferencia; para lo cual, utiliza un cronómetro que permite medir el tiempo de procesamiento, devolviendo este resultado en milisegundos.

Colaboración

Con ***cMotorInferenciaParaleo***. Dentro del contexto de inferencia en arquitectura heterogénea, es desde un objeto de esta clase que se ejecuta el objeto de la clase

cMotorInferenciaGPU, para comparar con los procesos de inferencia de las otras arquitecturas.

D) Clase cMotorInferenciaParalelo

Las especificaciones técnicas de esta clase, se muestran en la figura 32.

Figura 32

Especificación de la clase cMotorInferenciaParalelo en tarjeta CRC

Clase: cMotorInferenciaParalelo	
Responsabilidades	**Colaboración**
Atributos: aHechos aReglas aNroReglas **Métodos:** ProcesarInferenciaEnArquitecturaHeterogenea	 cMotorInferenciaCPU cMotorInferenciaNUCLEOS cMotorInferenciaGPU

Responsabilidades

Esta clase es la responsable de procesar las reglas y aplicarlas sobre los hechos, para efectuar el proceso de inferencia en todas las arquitecturas convencionales presentes en el equipo de cómputo.

Atributos:

aHechos de tipo arreglo bidimensional de números reales. Representa a la base de hechos, con los datos normalizados a valores numéricos reales y discretizados para su procesamiento.

aReglas de tipo arreglo bidimensional de números reales. Representa a la base de reglas, con las condiciones de cada regla, normalizados a una estructura bidimensional, donde en cada fila se representa una regla con todas sus condiciones.

Métodos:

ProcesarInferenciaEnArquitecturaHeterogenea

Parámetros de entrada: Hechos de tipo arreglo bidimensional.

Reglas de tipo arreglo bidimensional.

NroReglas de tipo entero.

Parámetros de salida: Result_CPU de tipo arreglo unidimensional de enteros.

Result_NUCLEOS de tipo arreglo unidimensional de enteros.

Result_GPU de tipo arreglo unidimensional de enteros.

Tiempo_CPU de tipo entero largo.

Tiempo_NUCLEOS de tipo entero largo.

Tiempo_GPU de tipo entero largo.

Descripción

Procesa las reglas y aplica las condiciones de éstas a la base de hechos, para efectuar el proceso de inferencia en los diferentes tipos de procesadores; para lo cual, en cada tipo de procesador, utiliza un cronómetro que permite medir el tiempo de procesamiento, devolviendo este resultado en milisegundos.

Colaboración

Con ***cMotorInferenciaCPU***, ***cMotorInferenciaNUCLEOS*** y ***cMotorInferenciaGPU***. Un objeto de la clase cMotorInferenciaParalelo invoca a objetos de estas tres clases, para efectuar los procesos de inferencia, medir los tiempos de procesamiento y efectuar una comparativa para estimar la bondad en cada tipo de arquitectura.

4.1.2.4 Especificación Técnica de las Estructuras de Datos. Dada la relevancia del procesamiento de las reglas y los hechos en memoria, determinadas sobre todo por la

arquitectura GPGPU, es importante especificar las características de las estructuras de datos utilizadas para representar estos hechos y reglas.

4.1.2.4.1 Estructura de Datos de la Base de Hechos. La estructura de datos requerida para almacenar en memoria principal la base de hechos, se representa mediante una matriz bidimensional, similar a una tabla de base de datos convencional; donde cada columna representa un atributo y cada fila una instancia del tipo de objeto a la que modela la respectiva tabla de hechos.

La base de datos deductiva física puede tener varias tablas de hechos, cada tabla de hechos se debe llevar a memoria principal, realizando los procesos de transformación, sobre todo la normalización y/o discretización a valores numéricos de tipo real. Esta restricción o condicionante de transformar los datos a valores numéricos, se debe a que la gran capacidad de cómputo de la arquitectura GPGPU sólo es posible sobre valores numéricos. Los procesos de inferencia realizados en arquitecturas CPU y NÚCLEOS, pueden utilizar cualquier tipo de datos sin restricciones; sin embargo, de acuerdo con el propósito del presente trabajo, para efectuar los análisis comparativos se implementa los procesos de inferencia sobre el mismo tipo de estructura de datos. En consecuencia, la figura 33 ilustra la estructura de datos utilizada para albergar la base de hechos en memoria.

Figura 33

Estructura de datos para los Hechos, en arquitectura CPU y NÚCLEOS

	Atributo1	Atributo2	Atributo3	Atributo4	Atributo5
Tupla1					
Tupla2					
Tupla3					
Tupla4					
Tupla5					

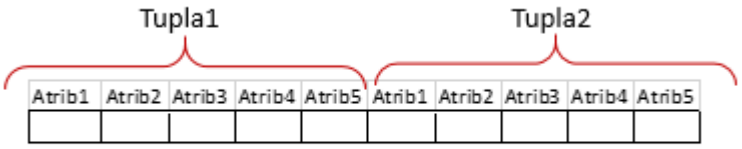
Matriz de Hechos para arquitectura CPU y NÚCLEOS

Nota. Elaboración propia.

Aunque también se debe considerar, que la arquitectura GPGPU impone aún una restricción adicional, sólo maneja estructuras de datos estáticas del tipo arreglo unidimensional. Por tanto, se realizará una transformación de la matriz bidimensional a un arreglo unidimensional para su procesamiento en los procesadores GPGPU, tal como se aprecia en la figura 34:

Figura 34

Estructura de datos para los Hechos, en arquitectura GPGPU



Matriz de Hechos para arquitectura GPGPU

Nota. Elaboración propia.

4.1.2.4.2 Estructura de Datos de la Base de Reglas. La estructura de datos requerida para almacenar en memoria principal la base de reglas, se representa también mediante una matriz bidimensional. Sin embargo, se debe considerar que una regla es una sentencia lógica expresada en "Forma Normal Conjuntiva - FNC"; además, el número de cláusulas conjuntivas varía para cada regla; en consecuencia, la matriz bidimensional tiene una semántica especial, cuyo detalle es el siguiente:

- Cada fila de la matriz almacena una regla, considerando todas las condiciones (cláusulas conjuntivas y disyuntivas) asociadas a ésta.
- La primera columna almacena el identificador de la regla.
- La segunda columna almacena el número de condiciones que tiene la regla.
- Las siguientes columnas almacenan la información de las condiciones, éstas son interpretadas en grupos de cuatro, donde cada condición se expresa en cuatro columnas, de acuerdo con la siguiente descripción:

- La columna 2+4*i almacena el atributo sobre el que está expresado la condición. El atributo se expresa en términos de número de columna en la matriz de hechos.
- La columna 3+4*i almacena el operador relacional discretizado a valores numéricos, tal los siguientes valores:

  = → 0, <> → 1, < → 2, <= → 3, > → 4, >= → 5.

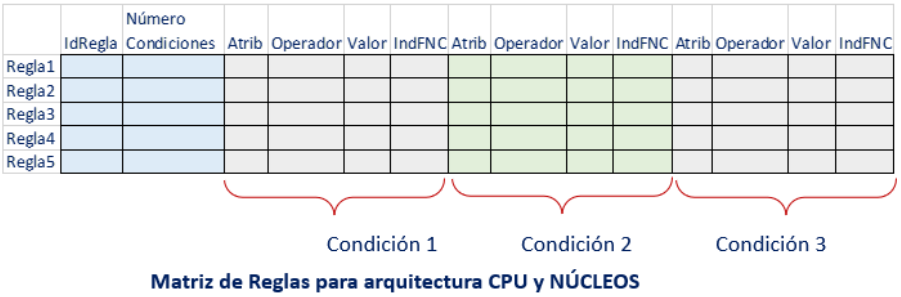
- La columna 4+4*i almacena el valor contra el que se compara el atributo, donde, el identificador i hace referencia al número de condición.
- La columna 5+4*i almacena el indicador del operador lógico aplicable a la siguiente condición. El valor 1 indica que se la cláusula está formada solo por una conjunción, mientras que un valor superior a 1 indica el número de disyunciones que se deben evaluar como parte de una conjunción.

o Finalmente, considerando que no todas las reglas tienen el mismo número de condiciones, entonces el número de columnas de la matriz bidimensional estará dado por el mayor número de condiciones de alguna regla multiplicado por cuatro más las dos primeras columnas.

Gráficamente esta estructura se aprecia en la figura 35:

Figura 35

Estructura matricial para las Reglas, en arquitectura CPU y NÚCLEOS



Nota. Elaboración propia.

Como se manifestó anteriormente, la arquitectura GPGPU sólo utiliza estructuras unidimensionales; en consecuencia, esta estructura es necesaria transformarla a una estructura lineal, con reglas una a continuación de la otra, en un arreglo unidimensional tal la figura 36.

Figura 36

Estructura unidimensional para las reglas en arquitectura CPU y NÚCLEOS

Nota. Elaboración propia.

4.1.2.5 Seudocódigo del Algoritmo del Motor de Inferencia Paralelo. En este punto no se considerará los algoritmos del motor de inferencia que correspondan a la arquitectura CPU, dado que éstos son convencionales y sólo son referenciales para propósitos de comparación en la estimación de la optimización de tiempos de procesamiento.

De acuerdo el propósito del presente trabajo, que trata de optimizar los tiempos de procesamiento de un motor de inferencia, utilizando algoritmos paralelos que se ejecuten en una arquitectura GPGPU; entonces, se debe considerar como estructurar el código para su funcionamiento en este contexto, para lo cual se debe tomar las siguientes consideraciones:

- La aplicación necesariamente debe ejecutarse en dos tipos de procesadores: Una parte en el procesador principal que es la CPU a la que se denomina "***Host***"; y otra parte en los procesadores GPGPU a los que se denomina "***Device***".
- La aplicación inicia su ejecución en el ***Host***, en base al modelo SISD de la taxonomía de Flynn; haciendo uso de los recursos del procesador principal, como: memoria, acceso a disco, etc. La parte del código que se ejecuta en el ***Host*** es el que se encarga de inicializar la aplicación, implementar las interfaces de usuario, así como de recuperar los hechos y

las reglas de la base de datos deductiva física, e interactuar con los procesadores de la GPGPU, tal como se explicó en el marco teórico, cuando se hizo referencia a las fases de los algoritmos en CUDA y OpenCL.

- Los procesos que se ejecutan en el ***Device***, corresponden a la parte del procesamiento en paralelo de cada unidad de datos. En el caso del presente proyecto, corresponden al procesamiento en paralelo de cada hecho de la base de hechos. Por tanto, se debe implementar módulos que se ejecuten en cada procesador del ***Device***, en base al modelo SIMD de la taxonomía de Flynn (más específicamente en base al modelo SIMT). La idea en este punto es que los hechos se procesen en paralelo, cada uno en un procesador de la GPGPU; haciendo uso de la memoria del Device.

A continuación, se muestra las unidades lógicas subyacentes en los algoritmos que deben ejecutarse tanto en el ***Host*** como en el ***Device***.

4.1.2.5.1 Seudocódigo del Algoritmo del "Device". En la figura 37 se ilustra un seudocódigo genérico al más alto nivel de abstracción, asociado al algoritmo del motor de inferencia paralelo, que se ejecuta en cada procesador de la GPGPU.

Este seudocódigo se puede implementar tanto en CUDA como en OpenCL. Para propósitos del presente trabajo, la implementación se realizó en OpenCL, cuyo código se adjunta en DVD al presente trabajo.

Figura 37

Seudocódigo del algoritmo del motor de inferencia en el "device"

```
MotorInferencia
INPUT: Vector Hechos, NroColumnasHechos, Vector Reglas, NroReglas
OUTPUT: Vector Resultado
INICIO
  /* Cada procesador GPU, procesará una fila de la base de Hechos
     para determinar a que regla corresponde o satisface dicha fila */
  NroGPU = RecuperarNumeroGPU_EnEjecucion()
  /* Recuperar la fila de hechos que verificará el procesador GPU */
  FilaHechos = RecuperarFilaHechosProcesadaPorGPU(NroGPU)
  /* La fila de hechos se contrasta contra las reglas,
     para determinar a que regla satisafce dicha fila de hechos */
  NroRegla = 0;
  FlagReglaOK = -1; // -- Aun no se encontró regla que satisfaga
  MIENTRAS((FlagReglaOK == -1) Y (NroRegla < NroReglas))
  INICIO
    /* Recuperar regla asociada a NroRegla */
    Regla = RecuperarRegla(NroRegla)
    /* Una regla se expresa mediante varias condiciones, entonces
       se debe verificar si la fila de Hechos satisface todas
       las condiciones de la regla */
    NroCondiciones = DeterminarNumeroCondicionesRegla(Regla)
    NroCondicion = 0
    FlagCondicion = 1; // -- Al inicio se asume que satisface condición
    MIENTRAS ((NroCondicion < NroCondiciones) Y (FlagCondicion == 1))
    INICIO
      /* Recuperar condición asociada a NroCondicion */
      Condicion = RecuperarCondicion(NroCondicion)
      /* Las condiciones están almacenadas en un único vector,
          organizados en una estructura que tiene tres partes:
         Número de la columna del atributo, el valor del atributo y
         el operador relacional; por tanto, se debe descomponer la
         condición en estos tres componentes */
      ColumnaAtributo, Valor, Operador = DescomponerCondicion(Condicion)
      /* Determinar si los valores de la fila de hechos, satisface la
         la condición */
      FlagCondicion = SatisfaceCondicion( FilaHechos, ColumnaAtributo,
                                          Valor, Operador)
      /* Siguiente condición */
      NroCondicion = NroCondicion + 1
    FIN
    SI (FlagCondicion == 1)
    INICIO
      /* Se encontró la regla a la que satisface la fila de Hechos;
         en consecuencia, asociar esa regla como resultado a la fila */
      Resultado[NroGPU] = RecuperarIdRegla(NroRegla)
      InterrumpirBucle
    FIN
    CASO CONTRARIO
      /* Procesar siguiente regla */
      NroRegla = NroRegla + 1;

  FIN
FIN
```

Nota. Código elaborado por el autor.

Esta es la lógica del algoritmo que se debe implementar y ejecutar en cada núcleo de la GPGPU, cualquiera que sea el lenguaje de programación utilizado.

Para ilustrar la implementación de las unidades lógicas relevantes del algoritmo anterior, se debe orientar el seudo código a algún lenguaje de programación de GPGPU. En el caso del presente trabajo, se ilustrará con un seudocódigo cercano a OpenCL, que es el lenguaje utilizado para la programación de la GPGPU.

Una de las unidades lógicas más relevantes, es la que determina la ejecución del algoritmo en paralelo. La GPGPU puede tener cientos o miles de procesadores, entonces todos los procesadores o núcleos ejecutan la misma lógica en paralelo, responsabilizándose cada uno de una fila de la matriz de hechos. ¿Cómo hacer saber a cada procesador de la GPU, qué fila debe procesar? Cada núcleo tiene un identificador que es un número entero que va desde cero hasta el número de procesadores que tiene la GPGPU. Entonces, se recupera este identificador y cada núcleo procesa la fila cuyo subíndice es igual al identificador. También se debe considerar, que cada núcleo se ejecuta en un hilo independiente. Este esquema obedece al modelo SIMT, el mismo conjunto de instrucciones (Single Instruction) se ejecuta en múltiples hilos (Multiple Threads). El algoritmo se muestra en la figura 38.

Figura 38

Algoritmo para recuperar el identificador de un núcleo GPGPU en OpenCL

```
RecuperarNumeroGPU_EnEjecucion()
INPUT:
OUTPUT: int NroNucleoGPU
INICIO
  /* El algoritmo del "device" se ejecuta en cada núcleo del
     GPU. Este módulo recupera el identificador del núcleo GPU en
     ejecución, de acuerdo a esta identificador se asocia el segmento de
     datos que debe procesar este núcleo.
     Este identificador se recupera mediante una función de OpenCL
  */
   return get_global_id(0); // worker principal

FIN
```

Nota. Código elaborado por el autor.

Otra sección relevante es la que corresponde a la recuperación de la fila que debe procesar cada núcleo. En la figura 39, se ilustra el seudocódigo correspondiente.

Figura 39

Algoritmo para recuperar la fila que debe procesar cada núcleo de GPU

```
RecuperarFilaHechosProcesadaPorGPU
INPUT: int NroGPU
OUTPUT: int FilaHechos  // -- Fila correspondiente a la matriz de hechos
INICIO
  /* Cada núcleo GPU procesa sólo una fila de la matriz de hechos.

     La matriz de hechos, se transfiere a la memoria del "device"
     como una estructura de datos unidimensional. Cada fila de la matriz
     es almacenada en el vector unidimensional una a continuación de
     otra.
     Este módulo recupera el inicio del segmento del vector
     unidimensional que corresponde a la fila que debe procesar el
     respectivo GPU.
  */
   int nroCols = nroColsV[0];
  return NroGPU * nroCols;
FIN
```

Nota. Código elaborado por el autor.

Este módulo recupera de la estructura unidimensional que almacena la base de hechos, el índice donde empieza los atributos que pertenecen a los hechos de la fila, que corresponde al núcleo GPU que procesa dicha fila. El algoritmo se detalla en la figura 40.

Figura 40

Algoritmo para recuperar la información correspondiente a cada regla

```
RecuperarRegla
INPUT:      int NroRegla
OUTPUT:     int Regla          // -- Regla correspondiente a NroRegla
INICIO
  /* Cada fila procesada por un núcleo GPU, debe ser
     Verificada si satisface alguna regla.
     La base de reglas es también una matriz estructurada en un
     arreglo unidimensional para ser procesado por la GPU
  */
  // -- Recuperar la dirección base de la regla
  int baseRegla = regla * (2+NroMaxCondicionesRegla*4);
  // -- Recuperar identificador de la regla
  int idRegla = (int)reglas[baseRegla];
  // -- Recuperar el número de Condiciones
  //    (columna:valor:operador:indFNC) de la regla
  int nroCondiciones = (int)reglas[baseRegla + 1];

FIN
```

Nota. Código elaborado por el autor.

Cada instancia de la base de hechos debe ser contrastada con las reglas de la base de reglas, para determinar la regla a la que satisface dicha instancia de la base de hechos. Se debe

considerar que cada regla está estructurada de una manera especial, donde los dos primeros atributos corresponden al identificador y al número de reglas; a su vez, cada regla tiene un determinado número de condiciones, sin embargo, en la estructura lineal se reserva espacio para el máximo número de condiciones permitidas en cada regla. El algoritmo se detalla en la figura 41.

Figura 41

Algoritmo para recuperar la información correspondiente a cada condición

```
DescomponerCondicion
INPUT:      int Condicion
OUTPUT:     int Columna, Valor, Operador, IndFNC // -- Componentes
INICIO
  /* Cada regla tiene condiciones; éstas, se deben verificar
     para determinar si son o no satisfechas, por los valores
     de los hechos de la fila actualmente procesada por un núcleo GPU.
     Las condiciones tienen una estructura especial en el arreglo
     Unidimensional de las reglas.
  */
  // -- Recuperar columna con la que se operará la condición
  int Columna = (int)reglas[baseRegla + 2 + Condicion * 4];
  // -- Recuperar valor con la que se comparará el valor de la columna
  float Valor = reglas[baseRegla + 3 + Condicion * 4];
  // -- Recuperar operador relacional que se aplicará en la comparación
  int Operador = (int)reglas[baseRegla + 4 + Condicion * 4];
  // -- Recuperar indicador Forma Normal Conjuntiva (Disyuntiva)
  int IndFNC = (int)reglas[baseRegla + 5 + Condicion * 4];
FIN
```

Nota. Código elaborado por el autor.

Cada regla consta de un conjunto de condiciones, donde cada condición o cláusula obedece a una nomenclatura especial, constituida por los elementos: Columna, Valor, Operador y IndFNC; donde, el valor de la columna debe ser operada contra el elemento Valor en base al operador; luego, la cláusula debe ser operada en forma de conjunción o disyunción, dependiendo del valor del elemento IndFNC (Indicador de la forma normal conjuntiva).

Si una fila de hechos satisface a todas las condiciones de la regla, entonces la fila de hechos satisface a la regla. El algoritmo se detalla en la figura 42.

Figura 42

Algoritmo para procesar una condición

```
SatisfaceCondicion
INPUT: float [] FilaHechos,
       int ColumnaAtributo, float Valor, int operador
OUTPUT: bool FlagCondicion
INICIO
  /* Verificar si la se satisface la condición */

  // -- Verificar que el hecho de la fila x y la columna,
       satisface el valor de la regla
  if (Operador == 0) // -- 0 significa =
     flagCondiciones = hechos[base + Columna] == Valor ? 1 : 0;
  else if (Operador == 1) // -- 1 significa !=
     flagCondiciones = hechos[base + Columna] != Valor ? 1 : 0;
  else if (Operador == 2) // -- 2 significa <
     flagCondiciones = hechos[base + Columna] < Valor ? 1 : 0;
  else if (Operador == 3) // -- 3 significa <=
     flagCondiciones = hechos[base + Columna] <= Valor ? 1 : 0;
  else if (Operador == 4) // -- 4 significa >
     flagCondiciones = hechos[base + Columna] > Valor ? 1 : 0;
  else if (Operador == 5) // -- 5 significa >=
     flagCondiciones = hechos[base + Columna] >= Valor ? 1 : 0;
FIN
```

Nota. Código elaborado por el autor.

4.1.2.5.2 Seudocódigo del Algoritmo del "Host". En la figura 43 se ilustra un seudocódigo genérico al más alto nivel de abstracción, asociado al algoritmo de inicialización y configuración de los procesos que deben ejecutarse en el procesador principal "host", para luego interactuar con los algoritmos que deben ejecutarse en cada procesador de la GPGPU "device".

Figura 43

Seudocódigo del algoritmo del motor de inferencia en el "host"

```
ProcesarInferenciaGPU
INPUT: TypeDev Dev_, float[,] hechos, int [] nroCols,
       float[,] reglas, int[] nroReglas, int[] NroMaxCondicionesRegla
OUTPUT: ref int[] result, long TiempoEjecucion
INICIO
  // -- Convertir matrices a vectores, porque el GPU procesa vectores
  ConvertirMatricesAVectores(hechos, reglas, hechosV, reglasV)
  // -- Activar cronómetro
  ActivarCronometro()
  // -- Inicializar dispositivo
  InicializarDispositivo(Dev_)
  // -- Copiar las estructuras de datos de los hechos y reglas
  //    de la memoria del "host" a la memoria del "device"
  CopiarVariables(hechosV, nroCols, reglasV, nroReglas, NroMaxCondicionesRegla)
  // -- Dimensionar número de hilos que procesarán cada fila de los hechos.
  //    Cada hilo se ejecutará en un núcleo GPU indpendiente
  int nroNucleos = ConfigurarNucleosGPU(hechos)
  // -- Ejecutar el algoritmo en el device
  EjecutarDevice(nroNucleos);
  // -- Recuperar los resultados de la memoria del "device" a la del "host"
  result = RecuperarResultados()
  // -- Detener cronómetro
  DetenerCronometro ()
  // -- Liberar buffer del GPU
  LiberarBufferDipositivo()
  // -- Devolver resultados
  retornar result, TiempoCronometro()
FIN
```

Nota. Código elaborado por el autor.

La implementación de las unidades lógicas identificadas en el algoritmo de la figura anterior varía en función al lenguaje de programación utilizado, así como a las librerías utilizadas para la gestión de la arquitectura CGPGU.

En el caso del presente trabajo se utilizó C# para la implementación en el lado del "Host", y las librerías de OpenCL para la implementación en el lado del "device".

4.1.3 Implementación del Motor de Inferencia Paralelo del Modelo Propuesto

El prototipo se implementó siguiendo estrictamente las clases identificadas y especificadas en la fase de diseño, considerando los siguientes aspectos:

4.1.3.1 Software Utilizado. El software requerido para la implementación del prototipo hace referencia al uso de un gestor de bases de datos y a los lenguajes de programación.

4.1.3.1.1 Sistema de Gestión de Base de Datos (SGBD). Considerando que en el presente trabajo sólo se pretende implementar un prototipo, se puede utilizar cualquier sistema de gestión de base de datos. La idea que subyace es que cada organización puede adaptar los conceptos y las propuestas vertidas en el presente trabajo, a su respectiva realidad; para lo cual, tendrán que efectuar adecuaciones a las diferentes herramientas software que utilizan.

En este contexto, en el presente trabajo se consideró el motor Microsoft SQL Server. Como software de interacción para la administración de la base de datos, se utilizó SQL Server Management Studio v17.9.1. Para la documentación pertinente referirse a Microsoft - SQL Server (2021).

4.1.3.1.2 Lenguajes de Programación. Por la razón manifestada en el párrafo anterior, también se puede emplear cualquier lenguaje de programación. Sin embargo, se utilizó los siguientes lenguajes:

- OpenCL para la implementación del Kernel, la parte del código que debe ejecutar cada procesador de la GPGPU. Básicamente, se implementó el seudocódigo descrito en la figura 35, con el rótulo de: *Seudocódigo del algoritmo del motor de inferencia en el "device"*. Cabe mencionar, que para la programación de la GPGPU se puede utilizar también el lenguaje CUDA; pero, este lenguaje sólo se utilizaría en infraestructuras que cuenten necesariamente con tarjetas NVIDIA. En ese sentido, OpenCL es más genérico.
- C Sharp para la implementación de las API's. Se implementó en base al concepto de librerías de enlace dinámico (DLL). El prototipo considera la implementación tanto

para la parte del proceso paralelo en arquitectura multinúcleo, así como la parte del proceso paralelo en arquitectura GPGPU. La sección de multinúcleo se programó con las librerías TPL (Task Parallel Library) de C Sharp. Para más detalle referirse a Microsoft - TPL (2021). La sección de GPGPU, se implementó con C Sharp la parte del código correspondiente al "Host" y con OpenCL la parte correspondiente al "Device". Para la parte de la interacción con el usuario, se implementó una capa de presentación circunscrita al mínimo requerido, con una tecnología del tipo "Windows Form".

4.1.3.2 Implementación de la Base de Datos. Los procesos de inferencia se pueden aplicar para satisfacer diferentes necesidades de información; en consecuencia, se pueden aplicar a diferentes conjuntos de datos y con reglas específicas para cada propósito.

En la implementación del prototipo, se consideró sólo un propósito; por tanto, se consideraron sólo tres tablas:

- taHechos: Tabla para almacenar la información de los hechos, sobre los que se realiza el proceso de inferencia.
- taReglas: Tabla para almacenar la información general de las reglas, con las que se realizará el proceso de inferencia.
- taReglasDetalle: Tabla para almacenar la información relativa a las condiciones que componen una regla.

A continuación, en las figuras 44 y 45 se muestran los scripts asociados a la base de datos:

Figura 44

Script para crear la base de datos

```
/* ************************************************************
     CREAR LA BASE DE DATOS PARA EL MOTOR DE INFERENCIA PARALELO
    ************************************************************ */
Create DATABASE BD_MotorInferenciaParalelo  -- Creates the Almacenes DataBase
on
   (NAME = BD_MotorInferenciaParalelo_mdf,    -- Primary data file
   FILENAME = 'D:\Data\BD_MotorInferenciaParalelo.mdf',
   SIZE = 100MB,
   FILEGROWTH = 10MB
   )
   LOG ON
   (NAME = BD_MotorInferenciaParalelo_Log,   -- Log file
   FILENAME = 'D:\Data\BD_MotorInferenciaParalelo.ldf',
   SIZE = 100MB,
   FILEGROWTH = 10MB
   )
go
```

Nota. Código elaborado por el autor.

Figura 45

Script para crear las tablas de la base de datos

```
/* ************************************************************
                CREAR LAS TABLAS DE HECHOS Y REGLAS
   ************************************************************ */

-- Tabla de hechos
create table taHechos(
  IdHecho int,
  Atributo1 Tipo1,
  Atributo2 Tipo2,
  ...

  primary key (IdHecho)
)
go

-- Tablas de reglas y detalle
create table taReglas(
  IdRegla int,
  Descripcion varchar(256) NULL,
  primary key (IdRegla)
)
go

create table taReglasDetalle(
  IdRegla int,
  IdDetalle int,
  Nombre_Atributo varchar(256) NULL,
  Operacion varchar(3) NULL,
  Valor numeric(17, 6) NULL,
  IndFNC int NULL,
  primary key (IdRegla, IdDetalle)
)
go
```

Nota. Código elaborado por el autor.

Para adecuar este prototipo a una situación específica, son estas tres tablas las que deben replicarse por cada propósito de inferencia. En la tabla de hechos se debe consignar todos los atributos y sus respectivos valores sobre los que se efectuará el proceso de inferencia. De manera similar, en la tabla de detalle de reglas, se debe consignar las cláusulas correspondientes a las sentencias lógicas; donde, en operación se debe poner los operadores relaciones y en el atributo IndFNC se debe especificar si una cláusula corresponde a la forma normal conjuntiva o disyuntiva.

4.1.3.3 Implementación del Código. Para la implementación del código correspondiente se creó una solución C Sharp, en la que se consideró la implementación de las API's, tanto para la API de interacción con la base de hechos y reglas, así como la API para los procesos de inferencia.

Para la parte del prototipo, se implementó otra solución C Sharp con una capa de presentación, que permite ilustrar la integración de las API's con aplicaciones de usuario final.

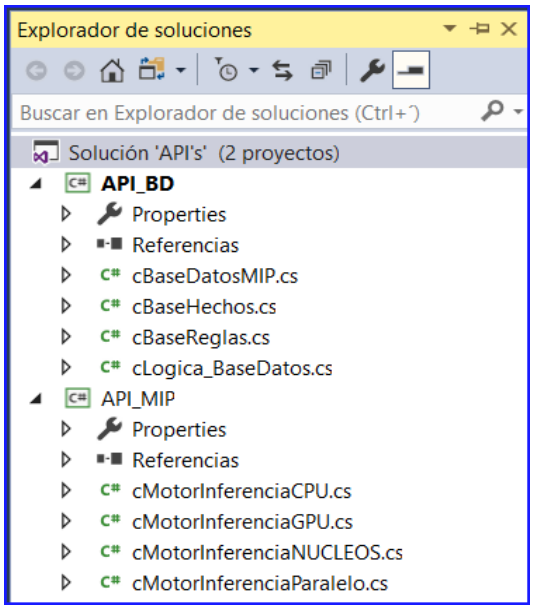
Seguidamente se hace una descripción sucinta de ambas implementaciones.

4.1.3.3.1 Implementación de las API's. La interfaz de programación de aplicaciones (API) se implementó en dos librerías de enlace dinámico (DLL).

A continuación, en la figura 46 se presenta el explorador de soluciones de Visual Studio, con la implementación de las API's.

Figura 46

Estructura de la solución que implementa las API's



Nota. Elaboración propia.

Primero se implementó el motor de inferencia paralelo en la ***API_MIP***, considerando la arquitectura heterogénea del computador. Dentro de esta librería se implementó las siguientes clases: *cMotorInferenciaCPU*, *cMotorInferenciaNUCLEOS* y *cMotorInferenciaGPU*; articulados por la clase *cMotorInferenciaParalelo*. La clase *cMotorInferenciaCPU* trabaja haciendo uso exclusivamente del procesador principal del computador; mientras que, *cMotorInferenciaNUCLEOS* trabaja haciendo uso de la arquitectura multinúcleo del computador; finalmente, la clase *cMotorInferenciaGPU* trabaja haciendo uso de la arquitectura GPGPU del computador.

Luego, se implementó las clases para el manejo de la base de datos del motor de inferencia paralelo en la ***API_BD***. Ésta, hace uso de las clases de la ***API_MIP*** e implementa a su vez las siguientes clases: cLogica_BaseDatos para la conexión a la base de datos; cBaseHechos, para gestionar los hechos de la base de datos; cBaseReglas, para gestionar la

base de reglas; finalmente cBaseDatosMIP, que es la que articula el funcionamiento de las otras clases.

4.1.3.4 Validación del Código Implementado. La parte neurálgica del prototipo implementado, lo constituyen las dos API's que subsumen la funcionalidad de la base de datos del motor de inferencia paralelo y los motores de inferencia en arquitectura heterogénea. En consecuencia, el proceso de validación debe efectuarse para garantizar el correcto funcionamiento de estas dos API's. Para tal propósito se implementó una base de datos de prueba y un módulo de validación, que se describen a continuación.

4.1.3.4.1 Implementación de la Base de Datos de Prueba. Se preparó datos de prueba, para los cuales se determinó manualmente los resultados del proceso de inferencia; contra los que se contrastó los resultados arrojados por el módulo de validación. Se consideraron las siguientes tablas:

Tabla de reglas, con 10 reglas asociadas a la identificación de tipos de ventas, que se muestra en la figura 47:

Figura 47

Reglas de la base de datos de prueba

IdRegla	Descripcion
1	Ventas gravadas en soles con boletas
2	Ventas gravadas en soles con facturas
3	Ventas no gravadas en soles con boletas
4	Ventas no gravadas en soles con facturas
5	Ventas con boletas anuladas
6	Ventas con facturas anuladas
7	Ventas de activo fijo en soles con boletas
8	Ventas de activo fijo en soles con facturas
9	Ventas con percepciones con facturas
10	Ventas con detracciones con facturas

Tabla de reglas_detalle, con las condiciones asociadas a cada regla de la tabla de reglas. Se implementó estas condiciones considerando los algunos tipos de venta que se presentan en las organizaciones. En la figura 48 se muestra un fragmento de ésta:

Figura 48

Condiciones de las reglas en la base de datos de prueba

	IdRegla	IdDetalle	Nombre_Atributo	Operacion	Valor	IndFNC
1	1	1	Cod_Tipo_Doc	=	3.000000	1
2	1	2	Cod_Moneda	=	0.000000	1
3	1	3	IGV	>	0.000000	1
4	1	4	ISC	=	0.000000	1
5	1	5	Modalidad_Pago	=	0.000000	1
6	1	6	Percepciones	=	0.000000	1
7	1	7	Retenciones	=	0.000000	1
8	1	8	Detracciones	=	0.000000	1
9	1	9	Cod_Cuenta_Venta	>	7000.000000	1
10	1	10	Indicador_Anulado	<>	1.000000	1
11	2	1	Cod_Tipo_Doc	=	1.000000	1
12	2	2	Cod_Moneda	=	0.000000	1
13	2	3	IGV	>	0.000000	1
14	2	4	ISC	=	0.000000	1
15	2	5	Modalidad_Pago	=	0.000000	1
16	2	6	Percepciones	=	0.000000	1
17	2	7	Retenciones	=	0.000000	1
18	2	8	Detracciones	=	0.000000	1
19	2	9	Cod_Cuenta_Venta	>	7000.000000	1
20	2	10	Indicador_Anulado	<>	1.000000	1
21	3	1	Cod_Tipo_Doc	=	3.000000	1
22	3	2	Cod_Moneda	=	0.000000	1
23	3	3	Ope_Inafectas	>	0.000000	2
24	3	4	Ope_Exoneradas	>	0.000000	1
25	3	5	Modalidad_Pago	=	0.000000	1

Tabla de hechos, con un conjunto pequeño de datos, seleccionados y adecuados convenientemente para satisfacer las diferentes condiciones de las reglas planteadas. Se preparó los datos de manera que cada regla sea satisfecha con dos de estas tuplas. En la figura 49 se muestra un fragmento de ésta:

Figura 49

Hechos en la base de datos de prueba

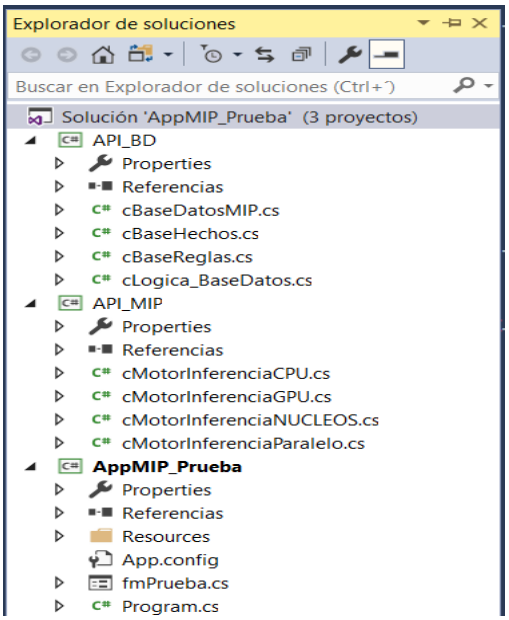
	IdHecho	Cod_Sucursal	Fecha_Emision	Cod_Periodo	Cod_Tipo_Doc	Nro_Serie	Nro_Doc	Doc_Identidad_Cliente	Nombre_Razon_Social_Cliente	Tipo_Cambio	Cod_Moneda
1	5065344	00	2021-02-16 17:32:35.010	2021-02	03	B002	1000006	66028056	Tirado Villarreal Lilia	1.000	PEN
2	5065356	00	2020-01-04 16:40:24.660	2020-01	03	B001	1000018	71110503	Acevedo González María	1.000	PEN
3	5066352	00	2021-02-24 09:19:20.947	2021-02	01	F001	1001014	26394959150	Nido Calderón SAC	1.000	PEN
4	5067191	00	2018-11-10 10:55:25.087	2018-11	03	B001	1001853	48318435	Arreola Villareal Quilla	1.000	PEN
5	5067711	00	2019-04-02 15:54:36.950	2019-04	03	B002	1002373	90229158	Melgar Zamora Gregorio	1.000	PEN
6	5068620	00	2018-03-02 16:16:44.497	2018-03	01	F001	1003282	23701570059	Salón de belleza Sánchez Carrión SAC	1.000	PEN
7	5068644	00	2018-08-30 11:29:47.327	2018-08	03	0011	1003306	40501674	Preciado Alvarado Ana Luisa	1.000	PEN
8	5068961	00	2018-08-30 11:28:06.123	2018-08	03	0011	1003623	27130382	Alcala Grijalva Sara	1.000	PEN
9	5083809	00	2018-07-05 12:25:42.110	2018-07	01	F001	1018471	32962131171	Centro dental Neptuno SAC	1.000	PEN
10	5111728	00	2020-07-22 13:59:43.633	2020-07	01	F001	1046390	79697496201	Nido Lucero	1.000	PEN
11	5114213	00	2020-09-30 15:46:42.517	2020-09	01	F006	1048875	25625563471	Ingenieros G&V SAC	1.000	PEN
12	5119238	00	2019-12-10 11:12:15.530	2019-12	01	F006	1053900	31150084864	Centro comercial Juárez EIRL	1.000	PEN
13	5136285	00	2019-04-22 10:01:43.590	2019-04	01	F001	1070947	10382950338	Centro de Salud Putumayo EIRL	1.000	PEN
14	5138162	00	2019-12-26 17:37:20.420	2019-12	01	F001	1072824	13188992251	Eventos Godínez SRL	1.000	PEN
15	5139260	00	2018-06-05 18:04:04.963	2018-06	01	F001	1073922	57831577233	Talleres Caylloma SRL	1.000	PEN
16	5139863	00	2018-04-03 13:24:45.947	2018-04	01	F001	1074525	92583729076	Botica M&P	1.000	PEN

4.1.3.4.2 Implementación del Módulo de Validación. Se implementó un módulo de validación, que haciendo uso de la base de datos de prueba, permitió validar el correcto

funcionamiento de las dos API's. En la figura 50, se muestra la estructura del módulo de prueba; donde, se tiene referenciadas las dos librerías a validar.

Figura 50

Estructura de la aplicación de validación

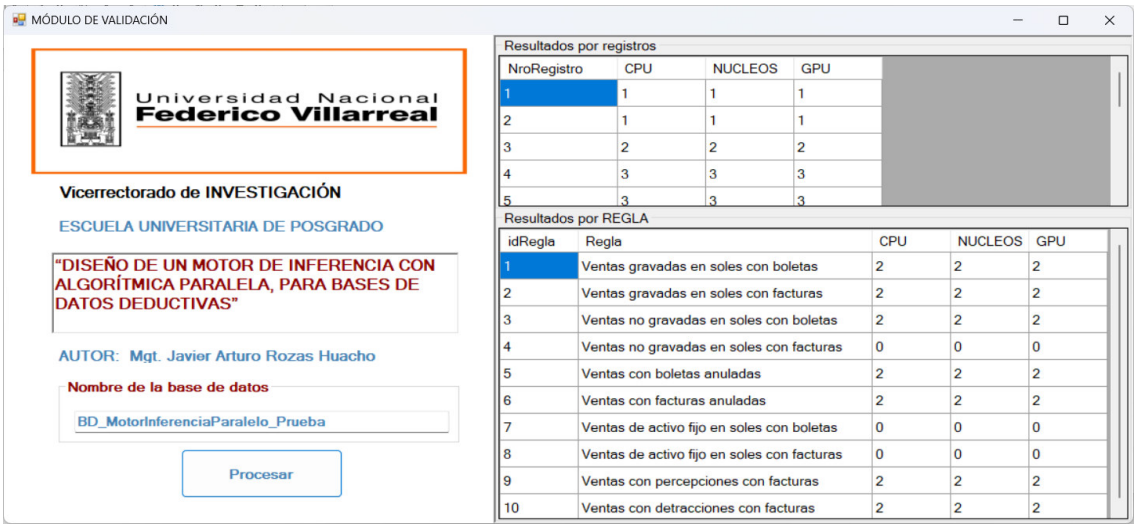


Nota. Elaboración propia.

Al ejecutar el módulo de validación, se obtuvo los resultados que se muestran en la figura 51:

Figura 51

Resultados de la ejecución del módulo de validación



Los resultados arrojados por el módulo de validación determinan el correcto funcionamiento del motor de inferencia paralelo implementado; dado que, los resultados arrojados corresponden a los resultados esperados de la base de datos de prueba. Como se puede apreciar en la primera ventana, cada registro de la base de hechos satisface a la misma regla en las tres arquitecturas planteadas. Así mismo, en la segunda ventana, cada regla es satisfecha con dos tuplas de la base de hechos, tal como se estructuró la base de datos de prueba.

Cabe mencionar que para las reglas 4, 7 y 8 no se consignaron datos, en consecuencia, son correctos los resultados mostrados.

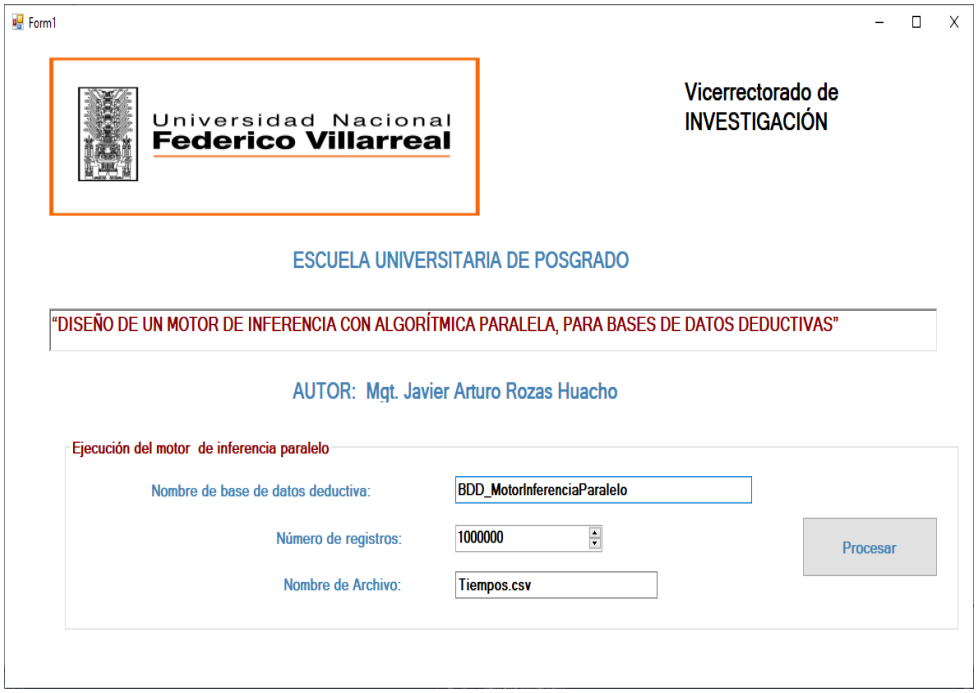
4.1.3.5 Aplicación Prototipo del Motor de Inferencia Paralelo. Dado que las organizaciones cuentan cada vez con mayores volúmenes de datos e información; se puede explotar estos datos con motores de inferencia en diferentes áreas de la organización.

Para la prueba del prototipo implementado, se debe seleccionar un caso de estudio que delimite el área de la organización donde se aplicará el motor de inferencia, así como definir la información que se desea inferir. Luego, se debe definir y tener acceso a la fuente de datos; para finalmente definir la base de datos, sobre todo la estructura de la tabla de hechos y las reglas con las que se realizará el proceso de deducción.

4.1.3.5.1 Implementación del Prototipo. La implementación del prototipo básicamente se circunscribe a la capa de presentación. Para lo cual, se diseñó un formato del tipo windows form que se muestra en la figura 52.

Figura 52

Pantalla principal de la aplicación



4.1.3.5.2 Selección de un Caso de Estudio. La aplicación de los procesos de deducción en los sistemas informáticos de gestión es diversa y amplia. Para efectos de este trabajo se tomará sólo un caso, el de determinar el tipo de venta al que corresponde cada venta, cuyo resultado permitirá por ejemplo el proceso de integración de los datos de ventas a la contabilidad. Se considera este caso, porque en toda organización dedicada a la comercialización de bienes y/o servicios, se requiere de este proceso.

Se debe considerar los siguientes criterios que justifican la necesidad de un proceso de deducción o inferencia:

- Cada organización tiene sus propias políticas de ventas, que pueden diferir incluso en organizaciones dedicadas al mismo rubro.

- Del mismo modo, cada contador diseña su propia estructura contable, acorde a la normatividad vigente y a los intereses de la organización. Este hecho determina los libros contables, así como el plan de cuentas a utilizar.
- La organización cuenta con un software integrado (ERP), con los subsistemas de ventas y contabilidad.
- Generalmente, un software integrado ya tiene implementado un módulo de integración a medida, acorde a la operativa de la organización. Este hecho implica que, cualquier adecuación o modificación en la normatividad, cambio en las políticas de gestión de la organización, cambio en el plan de cuentas, etc. Implicará la necesidad de la intervención de personal técnico informático; quienes, en coordinación con el personal contable-experto, deben intervenir el software para realizar dichos cambios.
- El problema planteado en el ítem anterior evidencia la dependencia de la organización de personal técnico; con los consecuentes costos y perjuicios que se podrían dar, por ejemplo, por el tiempo que se requiera para efectuar dichos cambios.

La propuesta del presente trabajo es propiciar una mayor automatización, de modo que los sistemas de información puedan realizar algunos procesos sin requerir la intervención de expertos humanos; así como la adecuación a nuevos escenarios, modificando simplemente algunas reglas en la base de datos deductiva de la organización.

4.1.3.5.3 Fuente de Datos. En el primer nivel de automatización todas las organizaciones cuentan con una base de datos transaccional. En este sentido, la fuente de los datos es la parte de la base de datos que corresponde al subsistema de ventas. Si bien cada organización tiene una base de datos con una estructura muy particular, éstas necesariamente deben de satisfacer las exigencias de la normatividad; por tanto, deben considerar toda la información requerida, por ejemplo, para la facturación electrónica. Por tanto, cualquiera que

sea la estructura de la base de datos transaccional, de ésta se debe extraer la información requerida para alimentar a la base de datos requerida para los procesos de inferencia.

Para propósitos del presente trabajo, se consideró los datos del subsistema de ventas de una base de datos transaccional real. Sin embargo, para resguardar la confidencialidad de estos datos, se reemplazó los atributos sensibles como nombres y documentos, con valores generados aleatoriamente, mediante una aplicación implementada para tal propósito denominada AppFaker.

Un aspecto relevante en esta fase de obtención de datos son las operaciones del proceso conocido como ETL (Extract-Transform-Load), donde:

La operación de extraer (Extract) es responsable de identificar las tablas y atributos específicos de la base de datos, de donde se recuperarán los datos requeridos para el dataset. En el caso del ejemplo, las tablas de ventas.

La operación de transformación (Transform) es responsable de uniformar los datos, efectuar las transformaciones necesarias para facilitar los procesos de inferencia en la base de datos deductiva. Dentro de estas transformaciones, se consideran, por ejemplo:

- La transformación de datos categóricos o nominales a datos numéricos. Teniendo en cuenta, sobre todo, que la arquitectura GPGPU sólo trabaja sobre datos numéricos.
- La discretización, es decir, convertir algunos atributos a valores discretos.
- La normalización, requerida en algunas circunstancias para transformar valores numéricos con un amplio rango de variabilidad, a un rango de 0 a 1.

La operación de carga (Load) es responsable de almacenar los datos previamente extraídos y transformados, en la base de datos del motor de inferencia paralelo.

4.1.3.5.4 Estructura de la Base de Datos del Motor de Inferencia Paralelo. A partir de los datos identificados en la base de datos transaccional, se construye la base de datos

requerida, para satisfacer las necesidades de cada proceso de inferencia; considerando las tablas de hechos, reglas y reglas detalle.

Para los propósitos del presente trabajo, en estas tablas se almacenará la información del subsistema de ventas y las reglas que guiarán el proceso de inferencia.

En la tabla 5, se ilustra la estructura de las tablas de hechos.

Tabla 5

Estructura de la tabla de hechos de la base de datos deductiva

NRO.	ATRIBUTO	Tipo
1	Cod_Sucursal	VARCHAR(15)
2	Fecha_Emision	DATETIME
3	Cod_Periodo	VARCHAR(15)
4	Cod_Tipo_Doc	VARCHAR(15)
5	Nro_Serie	VARCHAR(15)
6	Nro_Doc	VARCHAR(15)
7	Doc_Identidad_Cliente	VARCHAR(15)
8	Nombre_Razon_Social_Cliente	VARCHAR(15)
9	Tipo_Cambio	NUMERIC(5,3)
10	Cod_Moneda	VARCHAR(15)
11	Fecha_Vencimiento_Pago	DATETIME
12	Importe_Total	NUMERIC(17,6)
13	Importe_Total_Cargos	NUMERIC(17,6)
14	Descuentos_Globales	NUMERIC(17,6)
15	Doc_Concatenado	VARCHAR(15)
16	Modalidad_Pago	VARCHAR(15)
17	Indicador_Anulado	VARCHAR(15)
18	Cod_Tipo_Doc_Ref	VARCHAR(15)
19	Nro_Serie_Ref	VARCHAR(15)
20	Nro_Doc_Ref	VARCHAR(15)
21	Fecha_Emision_Ref	DATETIME
22	Cod_Periodo_Ref	VARCHAR(15)
23	Motivo	VARCHAR(15)
24	IGV	NUMERIC(17,6)
25	ISC	NUMERIC(17,6)
26	Otros_Imp	NUMERIC(17,6)
27	Ope_Gravadas	NUMERIC(17,6)
28	Ope_Inafectas	NUMERIC(17,6)
29	Ope_Exoneradas	NUMERIC(17,6)
30	Ope_Gratuitas	NUMERIC(17,6)
31	Percepciones	NUMERIC(17,6)
32	Retenciones	NUMERIC(17,6)

33	Detracciones	NUMERIC(17,6)
34	Bonificaciones	NUMERIC(17,6)
35	Total_Descuentos	NUMERIC(17,6)
36	FISE	NUMERIC(17,6)
37	Cod_Cuenta_Venta	VARCHAR(15)
38	Tipo_Afectacion_IGV	VARCHAR(15)
39	Observacion_Nro	VARCHAR(15)
40	Valor_Venta_Cuenta	NUMERIC(17,2)
41	Glosa	VARCHAR(15)
42	Cod_Tipo_Nota_Credito	VARCHAR(15)
43	Cod_Tipo_Nota_Debito	VARCHAR(15)
44	Cta_Nota_Credito_Ventas	VARCHAR(15)
45	Cta_Nota_Debito_Ventas	VARCHAR(15)
46	Nro	INT
47	Valor_Venta_Total	NUMERIC(17,6)
48	Nro_Items	INT
49	Observaciones	VARCHAR(15)
50	Porcentaje	NUMERIC(17,6)
51	Cod_Tasa_Detraccion	VARCHAR(15)
52	DPR	VARCHAR(15)
53	Tipo_Cambio_SUNAT	DECIMAL(9,4)

En la tabla 6 se ilustra la estructura de la tabla de reglas.

Tabla 6

Estructura de la tabla de reglas de la base de datos deductiva

NRO	ATRIBUTO	TIPO
	IDRegla	VARCHAR(15)
	Nombre_Regla	VARCHAR(256)

En la tabla 7 se ilustra la estructura de la tabla del detalle de las reglas.

Tabla 7

Estructura de la tabla de reglas_detalle

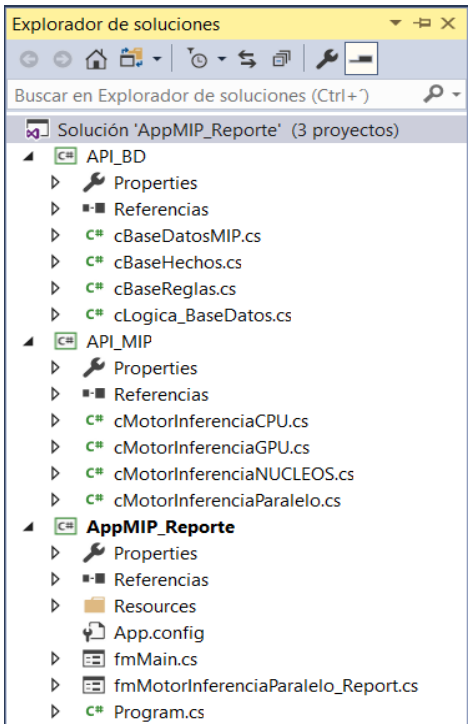
NRO	ATRIBUTO	TIPO
	IDRegla	VARCHAR(15)
	Nombre_Atributo	VARCHAR(256)
	Operación	INT
	Valor	NUMERIC(17,6)
	IndFNC	INT

Notar que la tabla de hechos se tiene que adecuar a los requerimientos de cada proceso de inferencia; mientras que, las otras dos tablas tienen una estructura fija válida para cualquier proceso de inferencia.

Las clases implementadas en la API_BD son las responsables de gestionar los datos de estas tablas. La estructura de la aplicación para mostrar los resultados, se muestra en la figura 53.

Figura 53

Estructura de la aplicación para mostrar los resultados



4.2 Disminución de Tiempos en el Proceso de Inferencia en Arquitectura Heterogénea

Como se mencionó a lo largo del trabajo, éste está orientado al uso de la infraestructura de cómputo heterogénea existente en el hardware de las organizaciones: CPU, Núcleos y GPGPU. El detalle de lo efectuado en cada tipo de procesador es el siguiente:

- Procesador CPU, en el que se realizó procesamiento secuencial. Los resultados obtenidos en este procesador se toman como base para las comparativas de disminución de tiempos.
- Procesadores multinúcleo, en la que se realizó uno de los tipos de procesamiento paralelo.
- Procesadores GPGPU, en las que se realizó el otro tipo de procesamiento paralelo.

En estos tres tipos de procesadores, se efectuó el proceso de inferencia de datos; para lo cual, se consideraron bases de hechos en el rango de las centenas de miles de tuplas.

4.2.1 Disminución de Tiempos en el Proceso de Inferencia en Arquitectura de Núcleos

Para efectuar el análisis de la disminución de tiempos en el proceso de inferencia en una arquitectura de Núcleos, se ejecutó el prototipo implementado en esta arquitectura y en la de CPU. Los resultados de la arquitectura CPU se toman como base comparativa para determinar los niveles de disminución de tiempos respecto a la arquitectura de Núcleos. Los resultados obtenidos se muestran en la tabla 8.

Tabla 8

Tiempos en arquitectura de Núcleos.

Número registros	Procesamiento en CPU (Milisegundos)	Procesamiento en NÚCLEOS (Milisegundos)
100000	37	19
200000	76	38
300000	122	38
400000	145	49
500000	178	58
600000	225	64
700000	251	73
800000	294	88
900000	340	98
1000000	357	106
1100000	417	119
1200000	428	121
1300000	479	129
1400000	509	134

1500000	530	142
1600000	565	149
1700000	590	169
1800000	654	169
1900000	676	182
2000000	742	192
2100000	750	192
2200000	770	202
2300000	831	218

Se puede apreciar una disminución en los tiempos de procesamiento paralelo respecto al secuencial. El análisis de la optimización y la eficiencia, se verán en el tópico 5.3.

4.2.2 Disminución de Tiempos en el Proceso de Inferencia en Arquitectura de GPU

Para efectuar el análisis de la disminución de tiempos en el proceso de inferencia en una arquitectura de GPGPU, también se ejecutó el prototipo implementado en esta arquitectura y en la de CPU. Nuevamente, los resultados de la arquitectura CPU se toman como base comparativa para determinar los niveles de disminución de tiempos respecto a la arquitectura de GPGPU. Los resultados obtenidos se muestran en la tabla 9.

Tabla 9

Tiempos en arquitectura de GPU.

Número registros	Procesamiento en CPU (Milisegundos)	Procesamiento en NÚCLEOS (Milisegundos)
100000	37	3
200000	76	5
300000	122	8
400000	145	13
500000	178	13
600000	225	17
700000	251	18
800000	294	22
900000	340	23
1000000	357	25
1100000	417	29
1200000	428	30
1300000	479	32
1400000	509	36

1500000	530	36
1600000	565	41
1700000	590	43
1800000	654	44
1900000	676	50
2000000	742	50
2100000	750	52
2200000	770	52
2300000	831	55

Como se puede apreciar, los tiempos de procesamiento paralelo en una arquitectura GPGPU son aún mucho más bajos que los tiempos de procesamiento en una arquitectura secuencial. El análisis de la eficiencia y los niveles de optimización, se verán en el tópico 5.3.

4.3 Determinación de la Eficiencia de Optimización

Para determinar la eficiencia de la optimización, se considera la métrica *SpeedUp*, tal como indica la teoría de la determinación de la eficiencia de los algoritmos paralelos; para lo cual, se considera los tiempos de procesamiento obtenidos en la sección anterior.

4.3.1 Eficiencia de Optimización en Arquitectura Multinúcleo

En base a los resultados mostrados en la sección 4.2.1 se elaboró la tabla 10 que muestra la eficiencia de la arquitectura multinúcleo en función de la métrica *SpeedUp*, considerando siempre el rango de las centenas de miles de tuplas.

Tabla 10

Eficiencia en arquitectura multinúcleo

Número tuplas	Procesamiento en CPU	Procesamiento en NÚCLEOS	$SpeedUp = \frac{Tiempo\ secuencial}{Tiempo\ paralelo}$
100000	37	19	1.95
200000	76	38	2.00
300000	122	38	3.21
400000	145	49	2.96
500000	178	58	3.07
600000	225	64	3.52
700000	251	73	3.44
800000	294	88	3.34
900000	340	98	3.47
1000000	357	106	3.37
1100000	417	119	3.50
1200000	428	121	3.54
1300000	479	129	3.71
1400000	509	134	3.80
1500000	530	142	3.73
1600000	565	149	3.79
1700000	590	169	3.49
1800000	654	169	3.87
1900000	676	182	3.71
2000000	742	192	3.86
2100000	750	192	3.91
2200000	770	202	3.81
2300000	831	218	3.81
Promedio	**433.30**	**119.52**	**3.63**

Considerando que la métrica *SpeedUp* indica que un algoritmo paralelo es eficiente respecto al algoritmo secuencial, sí éste tiene un valor mayor a 1. Entonces, en este caso, como se puede apreciar la arquitectura paralela del tipo multinúcleo en promedio se ejecuta con una eficiencia de 3.63 veces más rápido que en una arquitectura secuencial.

4.3.2 Eficiencia de Optimización en Arquitectura GPU

En base a los resultados mostrados en la sección 4.2.2 también se elaboró la tabla 11 que muestra la eficiencia de la arquitectura GPU, también en base a la métrica *SpeedUp*, considerando siempre el rango de las centenas de miles de tuplas.

Tabla 11

Eficiencia en arquitectura GPGPU

Número tuplas	Procesamiento en CPU	Procesamiento en GPU	$SpeedUp = \frac{Tiempo\ secuencial}{Tiempo\ paralelo}$
100000	37	3	12.33
200000	76	5	15.20
300000	122	8	15.25
400000	145	13	11.15
500000	178	13	13.69
600000	225	17	13.24
700000	251	18	13.94
800000	294	22	13.36
900000	340	23	14.78
1000000	357	25	14.28
1100000	417	29	14.38
1200000	428	30	14.27
1300000	479	32	14.97
1400000	509	36	14.14
1500000	530	36	14.72
1600000	565	41	13.78
1700000	590	43	13.72
1800000	654	44	14.86
1900000	676	50	13.52
2000000	742	50	14.84
2100000	750	52	14.42
2200000	770	52	14.81
2300000	831	55	15.11
Promedio	**433.30**	**30.30**	**14.30**

En este caso, como se puede apreciar la arquitectura paralela del tipo GPU en promedio se ejecuta con una eficiencia de 14.30 veces más rápido que en una arquitectura secuencial.

4.4 Proyección de Tiempos de Proceso de Inferencia en Arquitectura Heterogénea

Para proyectar e inferir a priori los tiempos de procesamiento para cualquier volumen de datos en una arquitectura heterogénea, es necesario, primero obtener funciones matemáticas que representen el comportamiento de los tiempos de ejecución en función del número de tuplas. Con este propósito, en esta sección primero se efectúa los procesos de ajuste de curvas mediante: regresión lineal, regresión polinómica de grado 2 y 3. Luego, en base al coeficiente de determinación se selecciona la función matemática que mejor se ajuste a los datos experimentales.

4.4.1 Proyección de los Tiempos de Inferencia en Arquitectura de Núcleos

Para efectuar los procesos de regresión, se adecuan los datos de la tabla de la sección 4.2.1 y se representa la variable dependiente en una escala de cientos de miles de tuplas. Sobre esta tabla es que se efectúa los diferentes procesos de regresión.

Para la proyección de los tiempos de procesamiento para cualquier número de tuplas en una arquitectura de núcleos, se ejecutó el prototipo implementado en esta arquitectura y en la de CPU. Los resultados en el CPU se toman como base comparativa para determinar los tiempos respecto a la arquitectura de Núcleos. Los resultados se muestran en la tabla 12.

Tabla 12

Tiempos en arquitectura de Núcleos para procesos de regresión.

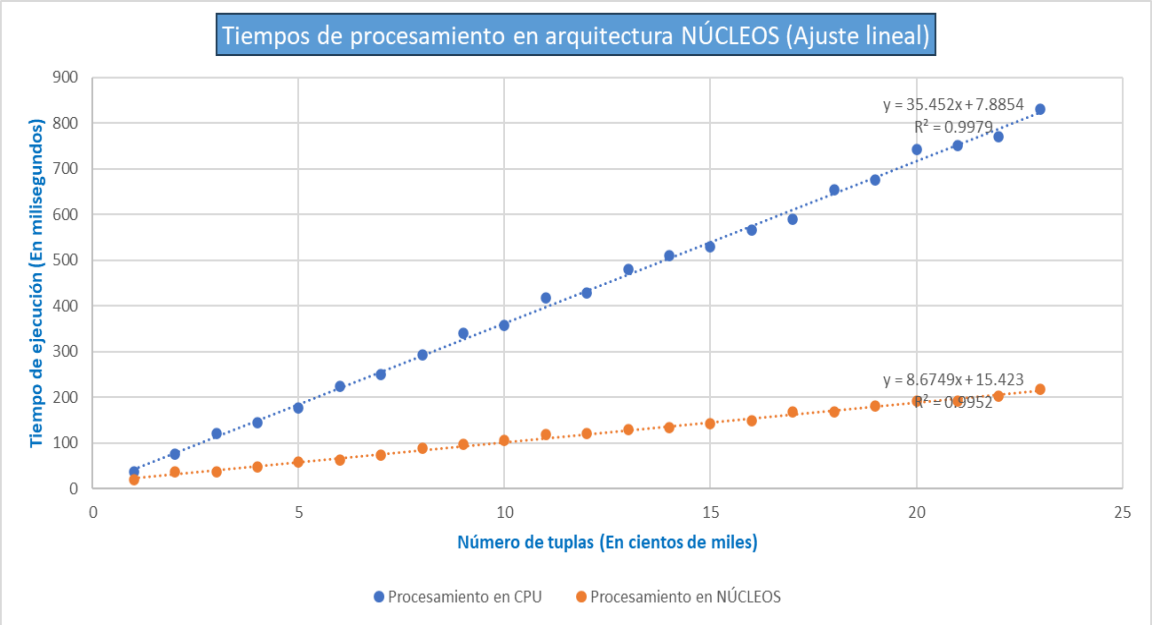
Número registros (En cientos de miles)	Procesamiento en CPU	Procesamiento en NÚCLEOS
1	37	19
2	76	38
3	122	38
4	145	49
5	178	58
6	225	64
7	251	73
8	294	88
9	340	98
10	357	106
11	417	119
12	428	121
13	479	129
14	509	134
15	530	142
16	565	149
17	590	169
18	654	169
19	676	182
20	742	192
21	750	192
22	770	202
23	831	218

Para proyectar los tiempos de procesamiento para diferentes volúmenes de datos, se procedió a graficar los resultados experimentales; luego, se efectuó regresión lineal y regresión polinomial de grado 2 y 3, para finalmente determinar la función matemática que mejor se ajuste a estos datos experimentales. A continuación, se muestra los procesos de regresión para la arquitectura multinúcleo.

4.4.1.1 Regresión Lineal de Tiempos de Ejecución en Arquitectura de Núcleos. A continuación, en la figura 54 se muestra la gráfica correspondiente a un ajuste lineal de los tiempos de ejecución del procesador de Núcleos.

Figura 54

Tiempos de procesamiento en arquitectura de Núcleos con ajuste lineal



Los tiempos de ejecución se ajusta muy bien a una función matemática lineal, tal como se puede percibir en la gráfica. Para establecer tiempos de ejecución para cualquier número de tuplas, es conveniente ajustar a funciones matemáticas los tiempos del procesador de tipo CPU (base comparativa) y los tiempos del procesador de tipo Núcleos. Las funciones matemáticas como los coeficientes de determinación se muestran a continuación:

- Función matemática para el procesador de tipo CPU

$$y = 35.452x + 7.8854 \qquad R^2 = 0.9979$$

- Función matemática para el procesador de tipo Núcleos

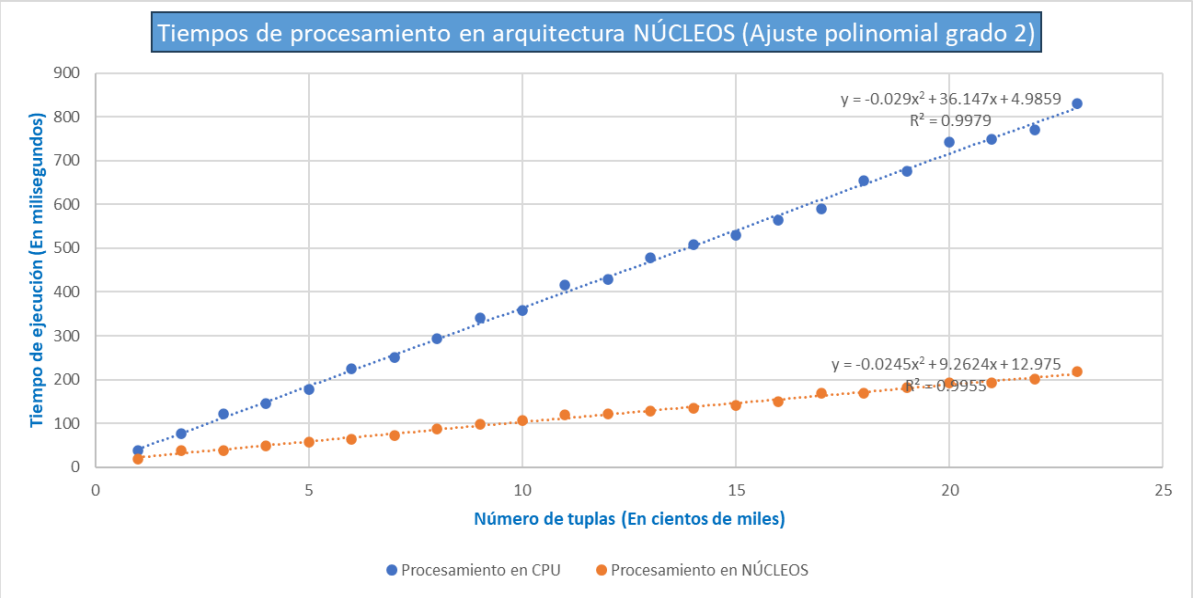
$$y = 8.6749x + 15.423 \qquad R^2 = 0.9952$$

El coeficiente de determinación $R^2 = 0.9952$ es bastante bueno; sin embargo, de todos modos, se efectúa el ajuste polinomial, para ver si aún se puede mejorar la bondad de esta regresión.

4.4.1.2 Regresión Polinomial Grado 2 de Tiempos en Arquitectura de Núcleos. A continuación, en la figura 55 se muestra la gráfica correspondiente a un ajuste polinomial de grado 2 de los tiempos de ejecución del procesador de Núcleos.

Figura 55

Tiempos en arquitectura de Núcleos con ajuste polinomial de grado 2



El nivel de optimización de los tiempos de ejecución se ajusta también muy bien a una función matemática polinomial de grado 2, tal como se puede percibir en la gráfica; aunque la diferencia con el ajuste lineal no es muy significativa. La función matemática como el coeficiente de determinación, se muestran a continuación:

- Función matemática para el procesador de tipo CPU

$$y = -0.029x^2 + 36.147x + 4.9859 \qquad R^2 = 0.9979$$

- Función matemática para el procesador de tipo Núcleos

$$y = -0.0245x^2 + 9.2624x + 12.975 \qquad R^2 = 0.9955$$

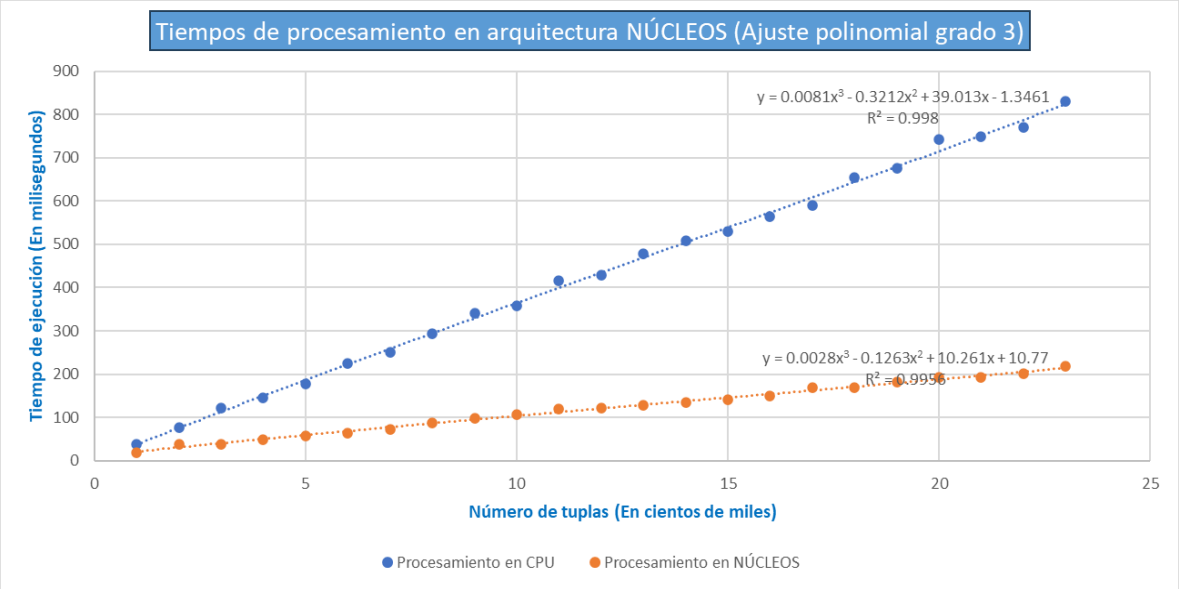
El coeficiente de determinación $R^2 = 0.9955$ es ligeramente superior al de la regresión lineal.

4.4.1.3 Regresión Polinomial Grado 3 de Tiempos en Arquitectura de Núcleos.

A continuación, en la figura 56 se muestra la gráfica correspondiente a un ajuste polinomial de grado 3 de los tiempos de ejecución del procesador de Núcleos.

Figura 56

Tiempos en arquitectura de Núcleos con ajuste polinomial de grado 3



El nivel de optimización de los tiempos de ejecución se ajusta también muy bien a una función matemática polinomial de grado 3, tal como se puede percibir en la gráfica; aunque, también la diferencia con el ajuste lineal y el ajuste polinomial de grado 2 tampoco son muy significativas. La función matemática como el coeficiente de determinación, se muestran a continuación:

- Función matemática para el procesador de tipo CPU

$$y = 0.0081x^3 - 0.3212x^2 + 39.013x - 1.3461 \qquad R^2 = 0.9979$$

- Función matemática para el procesador de tipo Núcleos

$$y = -0.0028x^3 - 0.1263x^2 + 10.261x + 10.77 \qquad R^2 = 0.9956$$

Las diferencias no son muy significativas en las regresiones de tipo lineal y polinomial tanto de grado 2 como el de grado 3; sin embargo, el coeficiente de determinación R cuadrado es ligeramente más preciso en la regresión polinomial de grado 3; en consecuencia, se podría utilizar ésta para calcular los tiempos de procesamiento para cualquier otro valor del número de tuplas.

4.4.2 Proyección de los Tiempos de Inferencia en Arquitectura de GPGPU

Para efectuar los procesos de regresión, se adecuan los datos de la tabla 5.2.2 y se representa la variable dependiente en una escala de cientos de miles de tuplas. Sobre esta tabla es que se efectúa los diferentes procesos de regresión.

Para el análisis de los tiempos de procesamiento de inferencia en una arquitectura de GPGPU, se ejecutó el prototipo implementado en esta arquitectura y en la de CPU. Los resultados de la arquitectura CPU se toman como base comparativa para determinar los tiempos de procesamiento respecto a la arquitectura de GPGPU. Los resultados obtenidos se muestran en la tabla 13.

Tabla 13

Tiempos en arquitectura de GPGPU para procesos de regresión.

Número registros (En cientos de miles)	Procesamiento en CPU	Procesamiento en GPGPU
1	37	3
2	76	5
3	122	8
4	145	13
5	178	13
6	225	17
7	251	18

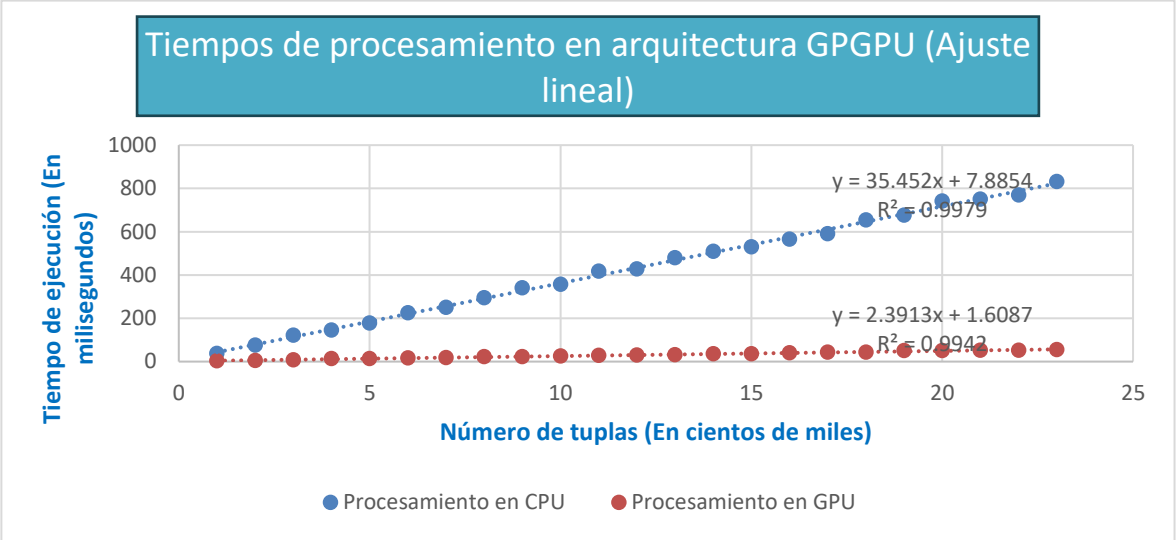
8	294	22
9	340	23
10	357	25
11	417	29
12	428	30
13	479	32
14	509	36
15	530	36
16	565	41
17	590	43
18	654	44
19	676	50
20	742	50
21	750	52
22	770	52
23	831	55

Para proyectar los tiempos de procesamiento para diferentes volúmenes de datos, se procedió a graficar los resultados experimentales; luego, se efectuó regresión lineal y regresión polinomial de grado 2 y 3, para finalmente determinar la función matemática que mejor se ajuste a estos datos experimentales. A continuación, se muestra los procesos de regresión para la arquitectura de GPGPU.

4.4.2.1 Regresión Lineal de Tiempos en Arquitectura de GPGPU. A continuación, en la figura 57 se muestra la gráfica correspondiente a un ajuste lineal de los tiempos de ejecución del procesador GPGPU.

Figura 57

Tiempos de procesamiento en arquitectura GPGPU con ajuste lineal



Los tiempos de ejecución se ajusta muy bien a una función matemática lineal, tal como se puede percibir en la gráfica. Las funciones matemáticas para el procesador de tipo CPU son las mismas; mientras que, las del tipo GPGPU y sus respectivos coeficientes de determinación se muestran a continuación:

- Función matemática para el procesador de tipo CPU

$$y = 35.452x + 7.8854 \qquad R^2 = 0.9979$$

- Función matemática para el procesador de tipo GPGPU

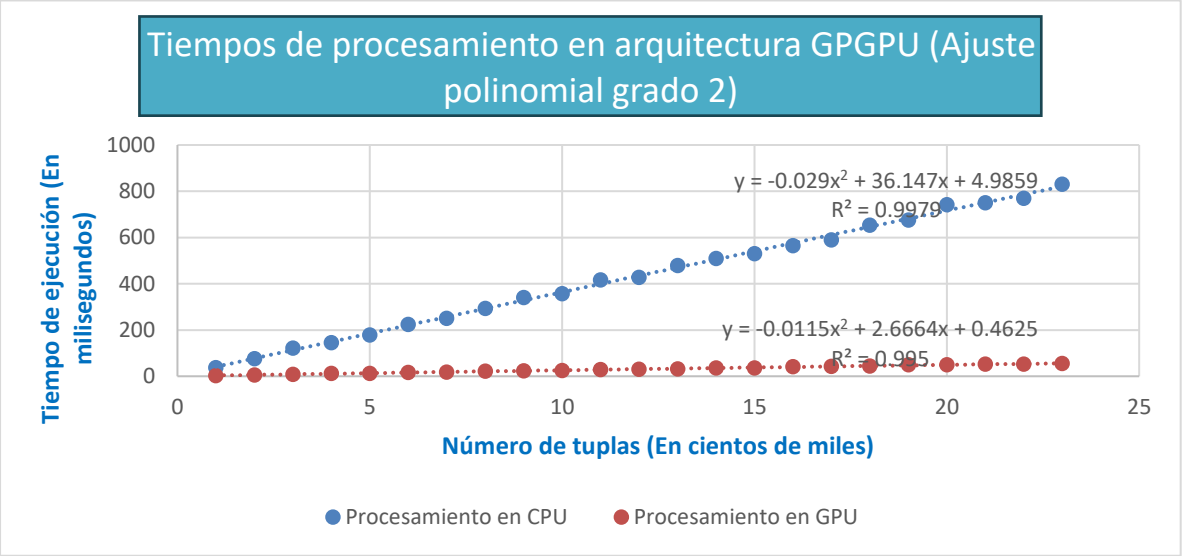
$$y = 2.3913x + 1.6087 \qquad R^2 = 0.9942$$

El coeficiente de determinación $R^2 = 0.9942$ es bastante bueno; sin embargo, de todos modos, se efectúa el ajuste polinomial, para ver si aún se puede mejorar la bondad de esta regresión.

4.4.2.2 Regresión Polinomial Grado 2 de Tiempos en Arquitectura de GPGPU. A continuación, en la figura 58 se muestra la gráfica correspondiente a un ajuste polinomial de grado 2 de los tiempos de ejecución del procesador GPGPU.

Figura 58

Tiempos en arquitectura de GPGPU con ajuste polinomial de grado 2



Los tiempos de procesamiento se ajusta también muy bien a una función matemática polinomial de grado 2, tal como se puede percibir en la gráfica; aunque la diferencia con el ajuste lineal no es muy significativa. La función matemática como el coeficiente de determinación, se muestran a continuación:

- Función matemática para el procesador de tipo CPU

$$y = -0.029x^2 + 36.147x + 4.9859 \qquad R^2 = 0.9979$$

- Función matemática para el procesador de tipo GPGPU

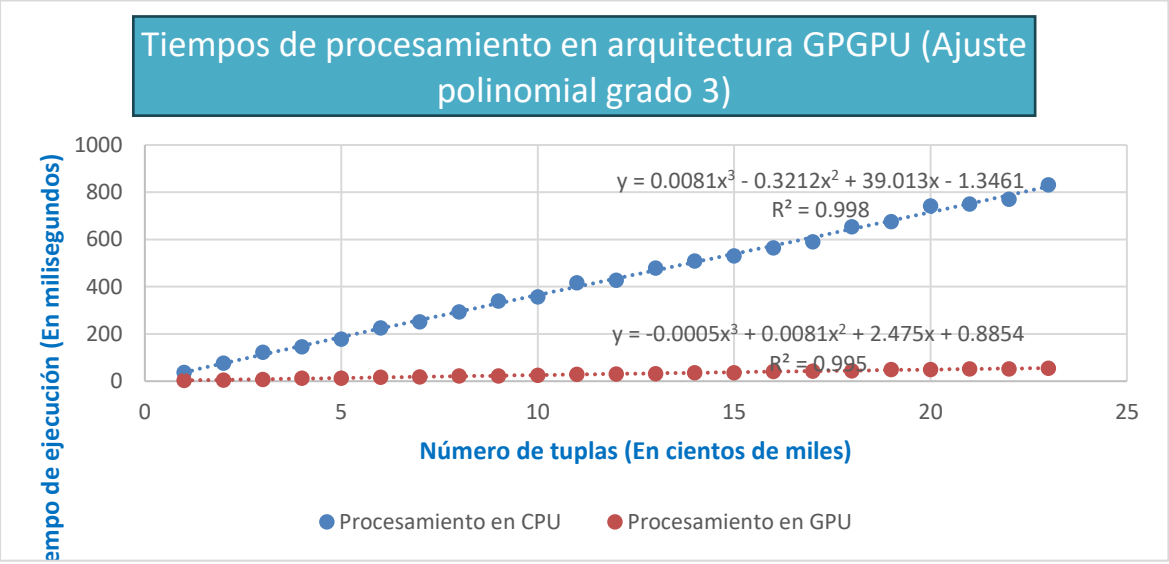
$$y = -0.0115x^2 + 2.6664x + 0.4625 \qquad R^2 = 0.995$$

El coeficiente de determinación $R^2 = 0.995$ es ligeramente superior al de la regresión lineal.

4.4.2.3 Regresión Polinomial Grado 3 de Tiempos en Arquitectura GPGPU. A continuación, en la figura 59 se muestra la gráfica correspondiente a un ajuste polinomial de grado 3 de los tiempos de procesamiento del procesador de GPGPU.

Figura 59

Tiempos en arquitectura de GPGPU con ajuste polinomial de grado 3



Los tiempos de procesamiento ajustados a un polinomio de grado 3, no presenta mejora alguna respecto al ajuste polinomial de grado 2. La función matemática como el coeficiente de determinación, se muestran a continuación:

- Función matemática para el procesador de tipo CPU

$$y = 0.0081x^3 - 0.3212x^2 + 39.013x - 1.3461 \qquad R^2 = 0.9979$$

- Función matemática para el procesador de tipo GPGPU

$$y = -0.0005x^3 + 0.0081x^2 + 2.475x + 0.8854 \qquad R^2 = 0.995$$

Como el coeficiente de determinación R cuadrado es igual al de la regresión polinomial de grado 2, entonces, se podría utilizar cualquiera de ellas para calcular los tiempos de procesamiento para cualquier otro valor del número de tuplas.

4.5 Validación de la Hipótesis

4.5.1 Validación de la Hipótesis General

La validación de la hipótesis se centra en demostrar que "*Si se usa un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU) se optimiza la deducción de información implícita en las bases de datos*". Efectivamente, con la información

de los cuadros de los dos anteriores acápites, se resume y se obtiene los promedios de los tiempos de procesamiento, que permiten validar la hipótesis general.

A continuación, en la tabla 14 se muestra los niveles de optimización logrados por el motor de inferencia paralelo en una arquitectura heterogénea.

Tabla 14

Optimización de tiempos de procesamiento en arquitectura secuencial y paralela

Tiempo secuencial	Tiempo paralelo			
	NÚCLEOS		GPGPU	
	Milisegundos	% Optimización	Milisegundos	% Optimización
433.3	119.52	72.42	30.3	93.01

Como se puede apreciar, utilizando la arquitectura de Núcleos se logra optimizar los tiempos de ejecución hasta en un 72.42%. Así mismo, con una arquitectura GPGPU se logra optimizar los tiempos de procesamiento hasta en un 93.01%.

4.5.2 Validación de las Hipótesis Específicas

La primera hipótesis específica, enunciada como "*Si se determinan las características relevantes que debe tener un motor de inferencia paralelo en una arquitectura heterogénea, éstas permitirán optimizar la deducción de información implícita en las bases de datos*", requirió que primero se efectúe el análisis de las características de un motor de inferencia paralelo en arquitectura heterogénea; como consecuencia de este análisis, se diseñó la propuesta del motor de inferencia paralelo orientado principalmente a la arquitectura GPGPU; siendo las características más relevantes:

- De las arquitecturas heterogéneas, la arquitectura GPGPU impone algunas restricciones: No admite algoritmos recursivos, así como las únicas estructuras de datos soportadas son las unidimensionales.
- Considerando que las expresiones lógicas inherentes a los procesos de inferencia pueden ser expresiones bastante complejas y con varios niveles de anidamiento; y que éstas, no se pueden resolver directamente con los algoritmos convencionales de inferencia en una arquitectura GPGPU; se determinó que estas expresiones lógicas deben estar expresadas en Forma Normal Conjuntiva (FNC), porque se evita el anidamiento de sentencias lógicas y son las pertinentes al uso de las estructuras de datos unidimensionales, requeridas en la arquitectura GPGPU.

  Se valida esta hipótesis específica, con la posterior implementación de este diseño y su aplicación en un prototipo, tal como se refiere en la sección 4.1.

La segunda hipótesis específica, enunciada como "*Si se usa un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU), se disminuye el tiempo de la optimización en la deducción de información implícita en las bases de datos*", se valida con lo expuesto en la sección 4.2. Como se vio, se logran disminuciones en los tiempos de procesamiento de un motor de inferencia paralelo. A continuación, en la tabla 15 se sintetizan dichos resultados:

Tabla 15

Disminución de tiempos de procesamiento en arquitectura heterogénea

Tiempo secuencial	Tiempo paralelo (En milisegundos)			
	NÚCLEOS		GPGPU	
	Ejecución	Disminución	Ejecución	Disminución
433.3	119.52	313.78	30.3	403

Como se puede apreciar, la disminución de tiempos de procesamiento es sustancial en las arquitecturas paralelas, siendo mucho mayor en la arquitectura GPGPU.

La tercera hipótesis específica, enunciada como "*Si se usa un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU), se incrementa la eficiencia de la optimización en la deducción de información implícita en las bases de datos*", se valida con lo expuesto en la sección 4.3. Como se vio, se logran incrementos en la eficiencia de procesamiento de un motor de inferencia paralelo.

La validación de esta hipótesis específica se basa en la métrica *SpeedUp*, que corresponde al concepto teórico utilizado en computación paralela.

Para la evaluación del rendimiento de los algoritmos paralelos en las diferentes arquitecturas, se utiliza la métrica *SpeedUp*, dada por la siguiente expresión:

$$S = \frac{Ts(n)}{Tp(n)}$$

Donde:

Ts = Tiempo de proceso secuencial

Tp = Tiempo de proceso en la arquitectura paralela

Los valores de esta métrica indican lo siguiente:

SpeedUp < 1 indica que el rendimiento del algoritmo paralelo es inferior al algoritmo secuencial.

SpeedUp = 0 indica que el rendimiento de ambos algoritmos es el mismo.

SpeedUp > 1 indica que el rendimiento del algoritmo paralelo es mejor que el rendimiento del algoritmo secuencial.

En consecuencia, la hipótesis será válida cuando el *SpeedUp* sea mayor a 1.

A continuación, en la tabla 16 se sintetizan los resultados logrados en los incrementos de eficiencia, primero para arquitectura multinúcleo y GPGPU:

Tabla 16

Eficiencia de procesamiento en arquitectura multinúcleo

Tiempo secuencial	Tiempo paralelo (NÚCLEOS)	$SpeedUp = \frac{Tiempo\ secuencial}{Tiempo\ paralelo}$
430.30	119.52	3.63

Luego, en la tabla 17 se sintetizan los resultados logrados en los incrementos de eficiencia en la arquitectura GPGPU:

Tabla 17

Eficiencia de procesamiento en arquitectura GPGPU

Tiempo secuencial	Tiempo paralelo (GPGPU)	$SpeedUp = \frac{Tiempo\ secuencial}{Tiempo\ paralelo}$
433.30	30.30	14.30

Como se puede apreciar, es sustancial el incremento de la eficiencia utilizando arquitectura GPGPU.

La cuarta hipótesis específica, enunciada como "*Si se usa un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU), se proyecta los tiempos de procesamiento para cualquier volumen de datos, en la deducción de información implícita en las bases de datos*", se valida sólo si es factible proyectar los tiempos de procesamiento para cualquier volumen de datos; esto es posible si se cuenta con funciones matemáticas que permitan efectuar dichas proyecciones.

En las secciones 4.4.1 y 4.4.2 se efectuó los respectivos procesos de regresión, habiéndose determinado que las funciones matemáticas que mejor representan los procesos de inferencia son del tipo polinomial de grado 3. Dichas funciones matemáticas son:

- Función matemática para el procesador de tipo GPGPU

$$y = -0.0005x^3 + 0.0081x^2 + 2.475x + 0.8854 \qquad R^2 = 0.995$$

- Función matemática para el procesador de tipo GPGPU

$$y = -0.0005x^3 + 0.0081x^2 + 2.475x + 0.8854 \qquad R^2 = 0.995$$

Donde, los coeficientes de determinación R cuadrado en ambos casos, indican que estas funciones matemáticas son bastante representativas del comportamiento de los tiempos de procesamiento de inferencia en una arquitectura heterogénea; por tanto, se pueden utilizar para efectuar proyecciones para diferentes volúmenes de datos.

V. DISCUSIÓN DE RESULTADOS

Los resultados obtenidos en el presente trabajo ponen en relevancia la importancia de la computación paralela en la implementación de motores de inferencia, que proporcionen información en línea en el orden de los milisegundos. Esta característica permite subsumir módulos de motores de inferencia paralelos, en las aplicaciones de gestión administrativa convencionales; añadiendo en consecuencia, capacidad de explicitar información implícita subyacente en: data bases convencionales, data warehouses o data lakes; que coadyuvarán a una mayor automatización y mejor gestión de los procesos tomando decisiones en tiempo real.

Una de las características más relevantes de los motores de inferencia paralelo en arquitectura heterogénea, es el uso de la Forma Normal Conjuntiva (FNC) para formalizar las expresiones lógicas, así como, facilitar su posterior procesamiento con estructuras de datos unidimensionales con procesadores del tipo GPGPU. El uso de estos motores de inferencia conlleva a mejorar las funcionalidades del software de gestión administrativa; con una sustancial disminución de los tiempos de procesamiento en los procesos de inferencia, lográndose reducir los tiempos de procesamiento en rangos que van desde más del 70% hasta más del 90%.

También se pone de relieve, que esta disminución de tiempos incrementa la eficiencia de estos procesos; evidenciándose, que la arquitectura GPGPU logra eficiencias de hasta más de 14 veces respecto al procesamiento secuencial; mientras que la arquitectura multinúcleo logra eficiencias de hasta más de 3 veces respecto al procesamiento secuencial.

Las eficiencias logradas corroboran las obtenidas en Gowanlock et al. (2021), se logra optimizar la eficiencia utilizando simultáneamente la arquitectura CPU/GPU.

Los resultados obtenidos corroboran también lo manifestado por Li H. et al. (2019), las capacidades de procesamiento de las GPU que van por encima de los 10 TFLOPS, permiten optimizar no sólo procesos de consulta, sino, también procesos de inferencia.

Respecto a la propuesta de Hu et al. (2020) en el uso de las GPU para la optimización de consultas, también es válido para la optimización de procesos de inferencia en bases de hechos y reglas.

En el trabajo se utilizan datos de bases de datos transaccionales, las que se adecúan a bases de hechos, lo cual también corrobora lo planteado por Lee et al. (2021), en la mejora de rendimiento tanto en sistemas de tipo OLTP y OLAP.

Adicionalmente, se puede manifestar que la arquitectura GPGPU es mucho más eficiente que la arquitectura multinúcleo; sin embargo, en este punto se debe considerar que la arquitectura GPGPU solo se puede aplicar sobre datos numéricos; mientras que, la arquitectura multinúcleo sobre cualquier tipo de datos; en consecuencia, no en todos los procesos de inferencia se pueden utilizar arquitectura GPGPU.

Respecto a efectuar proyecciones a priori sobre diferentes volúmenes de datos sin necesidad de efectuar el proceso experimental, que permitan obtener estimaciones de los tiempos que implicaría la ejecución de los procesos de inferencia de información implícita, es posible, dado que las funciones matemáticas que posibilitan estas estimaciones son representaciones bastante aproximadas de los datos. Se logra en ambos tipos de arquitectura, regresiones con coeficientes de determinación superiores a 0.99.

También enfatizar, que la expectativa que se tiene en el software de gestión administrativa va más allá de sólo la mecanización de procesos manuales, se espera que éste sea capaz de realizar tareas cada vez más complejas. Por ejemplo: Apoyo a la toma de decisiones en tiempo real, mayor automatización en procesos de integración, etc.

El prototipo ilustra la identificación de tipos de ventas en una organización, para lo cual se define una base de hechos y un conjunto de reglas; luego, mediante un motor de inferencia paralelo se efectúa la identificación indicada. Los resultados de este proceso de inferencia se podrían utilizar por ejemplo en los procesos de integración contable en línea, donde cada tipo de venta puede implicar un tipo de asiento distinto.

VI. CONCLUSIONES

- Se optimizó la deducción de información implícita de las bases de datos, mediante el análisis, diseño, implementación y uso de un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU), lo que se puede aplicar a software convencional de gestión administrativa.
- Se determinó las características del motor de inferencia paralelo en una arquitectura heterogénea, siendo las más relevantes, que las expresiones lógicas subsumidas dentro de las reglas, deben de estar expresadas en Forma Normal Conjuntiva (FNC); además, que la estructura de datos a utilizar debe ser unidimensional.
- Se disminuyó el tiempo de la optimización en la deducción de información implícita en las bases de datos, mediante la implementación y uso de un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU), lo que permitirá mayores niveles de automatización y dar mejor soporte a la toma de decisiones en tiempo real.
- Se incrementó la eficiencia de la optimización en la deducción de información implícita en las bases de datos, mediante la implementación y uso de un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU), lo que permitirá agregar más funcionalidades al software de gestión administrativa.
- Se determinó que las funciones matemáticas que permitirán determinar a priori los tiempos de procesamiento para cualquier volumen de datos son del tipo polinómico de grado tres. La validez de estas funciones está refrendada por los coeficientes de determinación con valores cercanos a 1.

VII. RECOMENDACIONES

- El presente trabajo se realizó sobre un computador de tipo Workstation, equipado sólo con una tarjeta gráfica del tipo dGPU de bajo costo y muy común en algunos equipos de hoy en día. Se recomienda efectuar estas pruebas computadores de tipo servidor y equipados con tarjetas gráficas del tipo iGPU y dGPU; así como sobre tarjetas más especializadas con varios miles de núcleos GPU, considerando tarjetas GPGPU de diferentes tipos, tal las Quadro de NVIDIA o RAEDON de AMD.
- El avance vertiginoso de la tecnología proporciona procesadores con más de 20 núcleos CPU, tal la serie de procesadores AMD Ryzen o la serie de Intel core i9. Se recomienda implementar estos motores de inferencia en arquitectura multinúcleo con el tipo de procesadores mencionados: luego comparar los valores de la métrica de SpeedUp con las que ofrecen las GPGPU de baja gama.
- Se recomienda también utilizar esta arquitectura de procesadores multinúcleo en la implementación de un motor de inferencia orientado a la taxonomía de Flynn del tipo MIMD. Dado que la arquitectura GPGPU sólo permite implementar la taxonomía del tipo SIMD, más específicamente la taxonomía SITM.
- Existen muchos contextos críticos donde se requiere de deducción de información implícita a partir de información explícita; por ejemplo, “data centers” con información estatal de alcance nacional, donde se exige latencias ultra bajas. En estos casos, utilizar arquitecturas GPGPU, puede implicar tiempos altos de procesamiento. Por tanto, se sugiere aplicar arquitectura basada en chips FPGA (Field Programmable Gate Array); considerando principalmente, que éstos contienen millones de celdas programables; que optimizarían grandemente los tiempos de procesamiento.

- Implementar sistemas expertos específicos, haciendo uso de motores de inferencia determinísticos en arquitectura heterogénea, tal lo planteado en el presente trabajo. Considerar con mayor énfasis la arquitectura GPGPU.

VIII. REFERENCIAS

Amdahl, G. (1967). *Validity of the single processor approach to achieving large scale computing capabilitie*. amdahl.dvi. www-inst.eecs.berkeley.edu//~n252/paper/Amdahl.pdf

Aranda, G. (2016). *Extensiones de bases de datos relacionales y deductivas: fundamentos teóricos e implementación*. Universidad Complutense de Madrid. https://eprints.ucm.es/id/eprint/38351/1/T37475.pdf

BBC News Mundo. (2021). *Qué es la cuarta revolución industrial (y por qué debería preocuparnos)*. https://www.bbc.com/mundo/noticias-37631834

Beck, Kent y Cunningham, Ward. (1989). *A Laboratory For Teaching Object-Oriented Thinking*. C2. http://c2.com/doc/oopsla89/paper.html#cards

Diccionario de la lengua española. (2023). *Lógica Formal y Matemática*. Real Academia Española. https://dle.rae.es/

Friedman, Thomas. (2006). *La tierra es plana* (1a ed.). Ediciones Martínez Roca, S.A. Paseo de Recoletos, 4. 28001.

Gaudiani, Adriana. (2012). *Análisis del rendimiento de algoritmos paralelos de propósito general en GPGPU. Aplicación a un problema de mallado de elementos finitos.* Sedici. http://sedici.unlp.edu.ar/bitstream/handle/10915/22691/Documento_completo__.pdf?sequence=1&isAllowed=y

Gowanlock, M., Fink, Z., Karsin, B. y Wright, J. (2021). *A study of work distribution and contention in database primitives on heterogeneous CPU/GPU architectures. Paper presented at the Proceedings of the ACM Symposium on Applied Computing*, 311-320. doi:10.1145/3412841.3441913

Grama, A., Gupta, A., Karypis, G. y Kumar, V. (2003). *Introduction to Parallel Computing* (2ª ed.). Addison Wesley.

Gustafson, J. (1988a). *Reevaluating Amdahl's Law*. http://www.johngustafson.net/pubs/pub13/amdahl.htm

Gustafson, J. (1988b). *Fixed Time, Tiered Memory, and Superlinear Speedup*. http://www.johngustafson.net/pubs/pub26/Superlinear.pdf

Harari, Yuval Noah. (2018). *21 lecciones para el siglo XXI* (1a. ed.). Pendin Random House Grupo Editorial.

Hernández Sampieri, R. y Mendoza Torres, C. (2018). *Metodología de la investigación las rutas cuantitativa, cualitativa y mixta* (1a. ed.) McGraw-Hill.

Hu, X., Xi, J., y Tang, D. (2020, June 15). Optimization for multi-join queries on the GPU. *IEEE Access*, 8, 118380-118395. https://doi.org/10.1109/ACCESS.2020.3002610

Instituto Nacional de Estadística e Informática [INEI]. (2018). *Tecnologías de Información y Comunicación en las Empresas*. https://www.inei.gob.pe/media/MenuRecursivo/publicaciones_digitales/Est/Lib1719/libro.pdf

Intel. (2019). *Familia de procesadores Intel Core serie X*. https://www.intel.la/content/www/xl/es/products/details/processors/core/x.html

Intel – ASIC. (2021). *Intel ASIC CDevices*. https://www.intel.la/content/www/xl/es/products/details/easic.html

Intel FPGAs Resource Center. (2021). *What is an FPGA? Programming and FPGA Basics - INTEL FPGAS*. https://www.intel.la/content/www/xl/es/products/details/fpga/resources/overview.html

Intel Xeon. (2019). *Intel Xeon Platinum 9200 Processor Performance*. https://www.intel.la/content/www/xl/es/benchmarks/server/xeon-scalable/platinum-9200-performance.html

Khronos – OpenCL. (30 de setiembre de 2020). *OpenCL Overview*. Khronos. https://www.khronos.org/opencl/

Lee, R., Zhou, M., Li, C., Hu, S., Teng, J., Li, D., y Zhang, X. (2021, July 1). The art of balance: A rateupdbtm experience of building a cpu/gpu hybrid database product. *VLDB Endowment*, 14(12), 2999-3013. https://doi.org/10.14778/3476311.3476378

Li, H., Tu, Y., y Zeng, B. (2019, April 16). Concurrent query processing in a GPU-based database system. *PLoS ONE*, 14(4), https://doi.org/10.1371/journal.pone.0214720

Microsoft – SQL Server. (2021). *Educational SQL resources*. https://docs.microsoft.com/es-ES/sql/sql-server/educational-sql-resources?view=sql-server-ver15

Microsoft – TPL. (2021). *Task-based asynchronous programming*. https://msdn.microsoft.com/es-es/library/dd537609(v=vs.110).aspx

Muller, Luis. (2011). *Programación concurrente Evaluación del rendimiento de Algoritmos Paralelos y/o Concurrentes*. Academia. https://www.academia.edu/9553807/UNIVERSIDAD_AMERICANA_Programaci%C3%B3n_Concurrente_Recopilaci%C3%B3n_de_teor%C3%ADa_referente_a_la_materia_Contenido

Naiouf, M. (2004). *Procesamiento Paralelo Balance de Carga Dinámico en Algoritmos de Sorting*. Universidad Nacional de La Plata. http://sedici.unlp.edu.ar/bitstream/handle/10915/2264/Documento_completo.pdf?sequence=1&isAllowed=y

NVIDIA. (2020). *Historia de NVIDIA una cronología de innovación*. https://www.nvidia.com/es-es/about-nvidia/corporate-timeline/

NVIDIA. (2020). *Programming Guide: CUDA Toolkit Documentation*. NVIDIA. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

Patterson, D. A., y Hennessy, J. L. (2014). *Computer Organization and Design* (5.ª edición). Morgan Kaufmann is an imprint of Elsevier.

Rentería, Vidales. (2015). *Tutorial para OpenCL en ANSI C, Java y C#*. Universidad de Sonora, Publicaciones del Departamento de Matemáticas.

Rucci, Enzo. (2017). *FPGAs: ¿los procesadores del futuro?* http://sedici.unlp.edu.ar/bitstream/handle/10915/64231/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y

Sahni, S y Thanvantri, V (1996). *Performance metrics: keeping the focus on runtime.* https://ieeexplore.ieee.org/document/481664

TechLib (2021). *Multi-Core*. https://techlib.net/definition/multi-core.html.

Thomasian, A. (2021). *Storage Systems: Organization, Performance, Coding, Reliability, and Their Data Processing* (p. 64). Pleasantville, Thomasian & Associates.

Tosini, Marcelo. (2015). *Rendimiento de sistemas paralelos*. Unicen. https://users.exa.unicen.edu.ar/catedras/arqui2/arqui2/filminas/Rendimiento%20de%20sistemas%20paralelos.pdf

Unicentro Cúcuta. (2019). *Mundo de hoy, aprender a aprender*. https://www.unicentrocucuta.com/uninoticias/mundo-de-hoy-aprender-a-aprender/

Universidad Europea de Madrid. (2017). *Introducción y Medidas de Rendimiento* (1.ª edición digital). Universidad Europea de Madrid, S.L.U. http://www.cartagena99.com/recursos/alumnos/apuntes/ININF1_M10_U1_T4_MT.pdf

Zanoni, Leandro. (2014). *Futuro inteligente* (1.ª edición digital). Recursos digitales.

GLOSARIO

API: (Application Programming Interface) Interfaz de programación de aplicaciones. Es un conjunto de normas y especificaciones que los programas de software pueden seguir para comunicarse entre sí.

Arquitectura: En el contexto informático se refiere a los componentes principales de un sistema y la forma como éstos interactúan y se coordinan para llevar a cabo el objetivo del sistema.

ASIC: (Aplication Specific Integrate Circuit) Circuito Integrado para Aplicaciones Específicas.

Bases de datos: Conjunto de datos de entidades u objetos y sus relaciones, almacenados con redundancia controlada, cuyo propósito es servir a una o varias aplicaciones y satisfacer necesidades de información de los usuarios. También debe almacenar la estructura y las restricciones.

CPU: (Central Processing Unit) Unidad central de procesamiento.

CRC: (Clase-Responsabilidad-Colaboración) Tarjetas utilizadas como parte de una metodología para el diseño de software orientado a objetos.

C Sharp: (C#) Lenguaje de programación de alto nivel, con el que se puede implementar software.

CUDA: (Compute Unified Device Architecture) Plataforma de programación paralela para tarjetas de tipo NVIDIA.

Data Lake: Es un repositorio de grandes volúmenes de datos estructurados y no estructurados, que provienen de diversos orígenes. Generalmente se almacenan sin efectuar

procesos de transformación de datos. Tiene como propósito fundamental, proveer la estructura que permita la exploración y el descubrimiento de patrones, que coadyuve al análisis de datos y a la toma de decisiones informadas.

Data Warehouse: En el contexto informático es una colección o almacén de datos que se utiliza como soporte a la inteligencia de negocios.

Data Mining: Es un conjunto de técnicas encaminadas a la extracción de conocimiento procesable, implícito en las bases de datos.

Device: Término asociado a los dispositivos de procesamiento paralelo sobre las que se implementan algoritmos paralelos.

dGPU: Unidad de procesamiento gráfico en tarjeta dedicada.

DLL: (Dynamic Link Library) Librería de enlace dinámico.

ETL: (Extract, transform and load) Extracción, transformación y carga. Corresponde a una etapa del proceso del Data Warehousing.

FNC: Forma Normal Conjuntiva.

FPGA: (Field Programmable Gate Array) Matriz de puertas programables en campo.

GPGPU: (General Purpose computing on Graphics Processing Units) Unidad de procesamiento gráfico de propósito general.

Host: Término asociado al procesador principal donde se ejecuta el código que centraliza y articula los algoritmos paralelos.

iGPU: Unidad de procesamiento gráfico en tarjeta integrada.

INEI: Instituto Nacional de Estadística e Informática.

Inteligencia de negocios: Conjunto de técnicas y herramientas para convertir datos en información y ésta en conocimiento, como soporte para la toma de decisiones estratégicas.

Inteligencia analítica: Conjunto de técnicas y herramientas matemáticas, estadísticas y de procesamiento de datos, orientadas a identificar patrones, tendencias y relaciones ocultas; que permitan esbozar planes estratégicos para la organización.

MIMD: (Multiple Instruction Multiple Data) Un tipo de arquitectura de cómputo.

MISD: (Multiple Instruction Single Data) Un tipo de arquitectura de cómputo.

Motor de inferencia: Se denomina "*motor de inferencias*" al componente que empleará la base de conocimiento para responder las consultas de los clientes. Este motor de inferencias deberá ser introducido por un experto informático y que será el que realice el mantenimiento del sistema.

NVIDIA: Empresa dedicada a la fabricación de unidades de procesamiento gráfico.

OLAP: (Online Analytical Proccesing) Procesamiento analítico en línea. Tecnología de inteligencia de negocios, orientada al análisis multidimensional de datos como soporte a la toma de decisiones.

OLTP: (Online Transactional Proccesing) Procesamiento de transacciones en línea. Técnica referida al registro de datos en las bases de datos transaccionales.

OMG: (Object Management Group) Organización Internacional responsible de la estandarización de tecnologías de la información, específicamente de las relacionadas con el paradigma orientado a objetos.

OpenCL: (Open Computing Language) Estándar abierto y multiplataforma para efectuar programación paralela en arquitectura heterogénea, independiente de la marca o tipo de procesador.

Paradigma: Conjunto de teorías, estándares y métodos que juntos representan una forma de organizar el conocimiento.

Script: Fragmentos de código o instrucciones de computador, utilizado principalmente en el contexto de las bases de datos.

Seudocódigo: Forma de expresar instrucciones para el computador, utilizando una mezcla de lenguaje natural y léxico de lenguajes de programación de computadoras.

SGBD: Sistema de Gestión de Base de Datos. Software mediante el cual se interactúa con las bases de datos.

SIMD: (Single Instruction - Multiple Data). Un tipo de arquitectura de cómputo.

SIMT: (Single Instruction - Multiple Threads). Arquitectura de cómputo que permite manejar múltiples hilos. Es la más apropiada para GPGPU.

SISD: (Single Instruction - Single Data). Un tipo de arquitectura de cómputo.

Sistema Experto: Software computacional capaz de comportarse como un experto humano en una determinada área.

SpeedUp: Métrica más usada para determinar la eficiencia de los algoritmos paralelos.

SQL: Lenguaje Estructurado de Consulta en sistemas de gestión de bases de datos relacionales (Structure Query Language).

SQL Server: Software de gestión de bases de datos de la empresa Microsoft.

FLOPS: (Floating point operations per second) Operaciones de coma flotante por segundo, ejecutadas por un procesador.

TIC: Tecnologías de la Información y la Comunicación.

TPL: (Task Parallel Library) Conjunto de módulos que permiten efectuar programación paralela en el lenguaje de programación C Sharp.

Tupla: Fila o registro de una base de datos.

UML: (Unified Modeling Language) Lenguaje Unificado de Modelado.

Visual Studio: Entorno de desarrollo integrado que soporta varios lenguajes de programación como C++, C#, Etc.

IX. ANEXOS

ANEXO 1. Matriz de consistencia

PROBLEMAS	OBJETIVO	HIPÓTESIS	VARIABLES, DIMENSIONES (D) E INDICADORES (I)	METODOLOGÍA
GENERAL ¿En qué medida se logra la optimización en la deducción de información implícita en las bases de datos, mediante el uso de un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU)? **ESPECÍFICOS**	**GENERAL** Optimizar la deducción de información implícita en las bases de datos, mediante el uso de un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU). **ESPECÍFICOS**	**GENERAL** Si se usa un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU) se optimiza la deducción de información implícita en las bases de datos.	**VARIABLES INDEPENDIENTES** **Motor de inferencia paralelo en una arquitectura heterogénea (Multinúcleo y GPGPU)** D-1. Tipo arquitectura I-1.1 Procesador de tipo multinúcleo o de tipo GPU I-1.2 Número de núcleos **Volumen de datos** D-2Tuplas de la base de hechos I-2.1 Número de registros **VARIABLES DEPENDIENTES**	**Enfoque:** Cuantitativo **Tipo:** Aplicada **Nivel:** Correlacional **Diseño** Experimental **Técnicas** Experimento

a) ¿Qué características debe tener un motor de inferencia paralelo en una arquitectura heterogénea, que permita optimizar la deducción de información implícita en las bases de datos? b) ¿En qué medida disminuye el tiempo de la optimización en la deducción de información implícita en las bases de datos, mediante el uso de un motor de	a) Determinar las características que debe tener un motor de inferencia paralelo en una arquitectura heterogénea, que permita optimizar la deducción de información implícita en las bases de datos. b) Disminuir el tiempo de la optimización en la deducción de información implícita en las bases de datos, mediante el uso de un motor de	**ESPECÍFICOS** a) Si se determina las características relevantes que debe tener un motor de inferencia paralelo en una arquitectura heterogénea, se optimiza la deducción de información implícita en las bases de datos. b) Si se usa un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU), se disminuye el tiempo de la optimización en la deducción de información	**Características del motor de inferencia paralelo en arquitectura heterogénea** D - 3. Características de motor de inferencia paralelo I-3.1. En una arquitectura multinúcleo I-3.2. En una arquitectura GPGPU **Optimización en la deducción de información implícita en las bases de datos** D - 4. Tiempo de optimización I-4.1. Disminución del tiempo de procesamiento en arquitectura multinúcleo I-4.2. Disminución del tiempo de procesamiento en arquitectura GPGPU.	**Instrumento** Ficha de observación estructurada, donde se consigna los tiempos de ejecución, así como la eficiencia de los procesos de optimización tanto de la arquitectura multinúcleo, así como de la GPGPU **Población:** La población está constituida por la totalidad de los datos de la base de datos y pueden ser aplicados a cualquier proceso de deducción de información implícita de las bases de datos. **Muestra** La muestra está constituida por la totalidad de los datos de ventas que serán utilizados en el proceso de deducción de información implícita de las bases de datos

inferencia paralelo en una arquitectura heterogénea? c) ¿En qué medida incrementa la eficiencia de la optimización en la deducción de información implícita en las bases de datos, mediante el uso de un motor de inferencia paralelo en una arquitectura heterogénea? d) ¿En qué medida se puede proyectar los tiempos de procesamiento para cualquier volumen	inferencia paralelo en una arquitectura heterogénea. c) Incrementar la eficiencia de la optimización en la deducción de información implícita en las bases de datos, mediante el uso de un motor de inferencia paralelo en una arquitectura heterogénea. d) Proyectar los tiempos de	implícita en las bases de datos. c) Si se usa un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU), se incrementa la eficiencia de la optimización en la deducción de información implícita en las bases de datos.	D - 5. Eficiencia de optimización I-5.1. Número de veces más rápido de procesamiento en una arquitectura multinúcleo respecto al procesamiento secuencial I-5.2. Número de veces más rápido de procesamiento en una arquitectura GPGPU respecto al procesamiento secuencial D – 6. Proyección de tiempos de procesamiento para cualquier volumen de datos (Previo ajuste a funciones matemáticas por regresión)	para identificar el tipo de venta al que pertenece cada venta. **Procesamiento y análisis de datos** • Preparación de los datos • Experimentación en diferentes arquitecturas con diversos volúmenes de datos. • Tiempos y eficiencia de optimización en los procesos de deducción de información implícita. • Métrica para medir la eficiencia: SpeedUp. • Regresión lineal y polinómica de grado 2 y 3, para luego a partir de las funciones matemáticas ajustadas, calcular tiempos de procesamiento para cualquier volumen de datos.

de datos, en la deducción de información implícita en las bases de datos, mediante el uso de un motor de inferencia paralelo en una arquitectura heterogénea?	procesamiento para cualquier volumen de datos, en la deducción de información implícita en las bases de datos, mediante el uso de un motor de inferencia paralelo en una arquitectura heterogénea.	d) Si se usa un motor de inferencia paralelo en una arquitectura heterogénea (multinúcleo y GPGPU), se proyecta los tiempos de procesamiento para cualquier volumen de datos, en la deducción de información implícita en las bases de datos.	I-6.1. Proyección de tiempos de procesamiento para cualquier volumen de datos en arquitectura multinúcleos I-6.1. Proyección de tiempos de procesamiento para cualquier volumen de datos en arquitectura GPGPU	• Métrica para medir la bondad de los ajustes de regresión: Coeficiente de determinación R^2