Escuela Politécnica Superior

Arquitectura Distribuida en la Nube para el cálculo específico de *software CAD/CAM* mediante procesamiento GPU.

Máster Universitario en Ingeniería Informática.

Trabajo Fin de Máster

Autor:
Víctor Sánchez Ribes.
Tutor/es:
Dr. Higinio Mora Mora.
Julio de 2019

Universitat d'Alacant
Universidad de Alicante

Resumen.

El paradigma de computación Cloud constituye una de las estrategias más innovadoras en cuanto a la adopción de Tecnologías de la Información por la sociedad. Las ventajas que proporciona permiten una mejora de la eficiencia y de reducción de costes, al tiempo que ofrece recursos y servicios accesibles a toda la sociedad. Sin embargo, su utilización masiva y generalización a nuevos sectores ha puesto de relieve problemas que deben ser abordados para satisfacer las demandas de calidad de servicio y mantener una adecuada relación entre rendimiento y coste. Por ejemplo, dotar de procesamiento especializado a determinadas aplicaciones.

Este proyecto tiene como objetivo construir un sistema de procesamiento Cloud Computing capaz de proporcionar procesamiento especializado en computación gráfica. Para realizar el proyecto se cuenta con una infraestructura compuesta por un servidor dotado de una tarjeta GPU avanzada. Se trata de especificar un procedimiento para que cuando una máquina local ejecute una tarea que lo requiera, la tarea sea externalizada a la nube y ejecutada remotamente por esta tarjeta.

Abstract.

The cloud computing paradigm is one of the most innovative strategies in terms of the adoption of Information Technology by society. The advantages it provides allow an improvement in efficiency and reduction of costs, while offering resources and services accessible to the whole society. However, its massive use and generalization to new sectors has highlighted problems that must be addressed to meet the demands for quality of service and maintain an adequate relationship between performance and cost. For example, provide specialized processing to certain applications.

This project aims to build a Cloud Computing processing system capable of providing specialized processing in graphic computing. To carry out the project, we have an infrastructure composed of a server with an advanced GPU card. It is about specifying a procedure so that when a local machine executes a task that requires it, the task is externalized to the cloud and executed remotely by this card.

Índice.

1. Introducción.

1.1. Objetivos del proyecto

Esta investigación nace de la necesidad de un sector de la industria por mejorar el servicio a sus clientes mediante el abaratamiento de los costes y aumentando la velocidad de fabricación de productos.

Este proyecto es una investigación que se propone estudiar el estado del arte de la computación en la nube, realizar una revisión de la arquitectura cliente – servidor y utilizar los avances existentes en ambos campos anteriormente citados junto con la elaboración de un *software* específico de diseño asistido por ordenador o *software CAD/CAM*, ampliamente utilizado en la industria y con mayor concreción en la industria del calzado vastamente instalado alrededor de esta Universidad.

Con esto se pretenden unos objetivos para poder elaborar una conclusión bien fundamentada para que este trabajo sirva como punto de partida para la posterior investigación en este campo que lleve a cabo la ejecución de forma eficiente y eficaz de este tipo de *software* ejecutando las funciones de mayor requerimiento de cálculo y de procesamiento especifico que se ejecutará en la unidad de procesamiento gráfico (*GPU*) mediante el lenguaje, propio de NVIDIA, llamado CUDA.

Por esto el objetivo principal de este trabajo es la creación de una primera arquitectura de *software CAD/CAM* que permita ejecutar las funciones de mayor requerimiento de procesamiento en servidores dedicados alojados en la nube con las ventajas que supone como la contratación de requisitos a medida.

Como objetivos secundarios la medición de los tiempos de procesamiento tanto de la ejecución de funciones de gran requerimiento de procesamiento en el procesador del ordenador (*CPU*) comparando con el tiempo de procesamiento de las mismas funciones en la unidad de procesamiento grafico como es la *GPU*, así como la medición de los tiempo necesarios en la ejecución en la arquitectura nube configurada en este trabajo no solamente de ejecución de cálculos sino que también influirán en este caso la latencia de la red y el tiempo tanto del envío como de la recepción de los cálculos. Y, como último objetivo, se emitirán las conclusiones dadas por los datos de la experimentación, para lo que se evaluarán los resultados y su posible incorporación real, que se llevará a cabo para la realización de esta investigación.

1.2. Motivación

Tras observar el desarrollo tecnológico en el campo de las tarjetas gráficas dedicadas y su potencia de cálculo superior a los procesadores para uso general como la *CPU* del ordenador se hace interesante la investigación en este campo para paliar o mejorar las necesidades de las empresas tanto desde el punto de vista económico, con la reducción de costes, como con la minimización de tiempos para realizar sus negocios.

Por esto además de la viabilidad de la propuesta, mi motivación personal recae en la posibilidad de aumentar esta investigación y poder presentar una tesis doctoral con la que conseguir presentar una arquitectura efectiva y poder implantarla en empresas para su uso real.

1.3. Definición del problema

Uno de los principales apartados necesarios para crear esta arquitectura es conocer el problema que se tiene que solucionar.
Actualmente, las empresas que utilizan software de tipo CAD/CAM, para diseño en 2 dimensiones como de 3 dimensiones, necesitan equipar a los departamentos de diseño principalmente, de equipos con altas prestaciones y *hardware* dedicado y especifico, normalmente de elevado precio y probablemente infrautilizados con lo que el desembolso es mayor.
Por esto, el problema recae investigación de la utilización óptima de la *GPU*, así como asegurar la paralelización de los cálculos. También es necesario conocer la forma de comunicación con un servidor para enviar y recibir tanto los cálculos como métricas para optimizar tanto la comunicación como el cálculo de forma óptima para disminuir los tiempos de comunicación tanto en redes móviles (*3G, 4G, 5G, GPRS*…) como redes alámbricas e inalámbricas.

1.4. Propuesta de solución

Tras reconocer el problema se propone una solución como una arquitectura distribuida para la ejecución de un programa con funciones de alto requerimiento computacional de forma distribuida. Para esto se propone una arquitectura de computación basada en la arquitectura Cliente - Servidor clásica remodelada para hacer frente a las exigencias encontradas y explicadas de forma más extendida y profunda en el punto 4.

2. Estado actual de la investigación.

2.1. Introducción al Cloud Computing.

Con el rápido desarrollo de las tecnologías de procesamiento y almacenamiento y el auge de Internet, la computación se ha convertido en un recurso más barato, ubicuo y con mayor potencia.

Esto es debido a un nuevo paradigma llamado, computación en la nube, mediante el cual los recursos y utilidades pueden ser ofrecidos a los usuarios que lo deseen a través de Internet en una modalidad bajo demanda, por la que se ofrecen recursos como sean necesarios [7].

La aparición de este nuevo paradigma ha hecho que multinacionales de la talla de Google, Amazon, Microsoft ahonden en este nuevo paradigma para ofrecer herramientas con mayor, si cabe, potencia, mayor fiabilidad y con un coste muy competente, así que las empresas empiezan a reformular sus negocios para aprovechar las ventajas que esta tecnología ofrece, como:

- No es necesaria una inversión inicial elevada, ya que se alquilan los recursos que se necesitan.
- Coste operacional muy bajo ya que se liberan los recursos no utilizados para ahorrar costes cuando la demanda de servicio es baja.
- Alta escalabilidad proporcionada si el negocio crece mediante la agrupación de grandes cantidades de recursos en un centro de datos.
- Fácil acceso al estar alojada en la nube y accesible por Internet con una cantidad elevada de dispositivos móviles.
- Riesgos empresariales y de mantenimiento reducidos puesto que al ser un servicio subcontratado a la nube el riesgo de fallo es menor por la experiencia de los proveedores de la infraestructura.

2.2. Arquitectura de Computación en la Nube.

En esta sección describiremos la capa de arquitectura, la capa de negocio y modelos operacionales del paradigma computación en la nube.

2.2.1. Modelo de capas.

La arquitectura de un entorno de la nube puede dividirse en 4 capas; capa de hardware, capa de infraestructura, capa de plataforma y capa de aplicación [1].

- Capa de hardware es la encargada de gestionar los recursos físicos de la nube, incluyéndose como recursos físicos: enrutadores, servidores, conmutadores y los módulos de batería y de ventilación.
- Capa de infraestructura, también conocida como capa de virtualización, es la encargada de crear conjunto de recursos de almacenamiento y de computación.

- Capa de plataforma consiste en el conjunto de sistemas operativos y *frameworks* utilizados que minimiza la carga del desarrollo de aplicaciones en entornos virtualizados.
- Capa de aplicación, finalmente, es la aplicación de la nube en sí. Estas aplicaciones pueden balancear la escalabilidad para ofrecer mejor rendimiento y disponibilidad a un mínimo coste.

2.2.2. Modelos de negocio.

El paradigma de computación en la nube utiliza el servicio bajo demanda por el que se proporcionan recursos según la demanda necesaria del usuario [4].

Actualmente, la nube ofrece servicios agrupados en 3 categorías; Software como Servicio (*SaaS*), Plataforma como Servicio (*PaaS*) e Infraestructura como Servicio (*IaaS*), como se muestra en la Figura 1, se añaden ejemplos de distintas herramientas según los recursos que se demandan:

- *Infraestructura como servicio (IaaS):* se refiere al aprovisionamiento de recursos bajo demanda.
- *Plataforma como servicio (PaaS):* se incluye el soporte de los sistemas operativos y los *frameworks* para el desarrollo de aplicaciones.
- *Software como servicio (SaaS):* todo aquello relacionado con las aplicaciones de Internet.

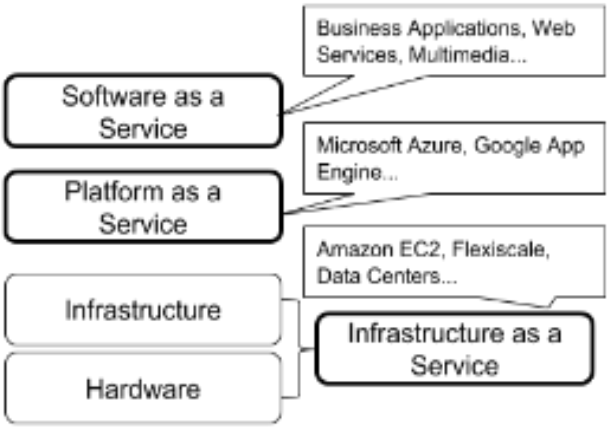


Fig 1. Modelos de negocio en la nube.

2.2.3. Tipología de Nube.

Para poder migrar la aplicación empresarial a un entorno en la nube es imprescindible el estudio de factores como el coste operacional, la fiabilidad o la seguridad ya que cada tipo de cloud ofrece unas especificaciones que repercutirán en el coste.

- *Nube Pública:* los proveedores de servicio ofrecen sus recursos al público en general. Tiene algunas ventajas como la inversión inicial, en infraestructura, nula, pero en cambio hay una falta de seguridad y control sobre los datos [6].

- *Nube Privada:* son los utilizados por únicamente una organización y gestionada por la misma organización o servicios externos. Ofrecen el mayor grado de control sobre la efectividad, la fiabilidad y la seguridad, aunque no consta de una inversión en infraestructura casi nula.

- *Nube Híbrida:* combinación de las dos anteriores. En este tipo se proporciona un mayor grado de control y seguridad sobre los datos de la aplicación en comparación con las nubes públicas mientras facilita la expansión bajo demanda. Por el contrario, se necesita separar muy detenidamente aquello que se ejecutará sobre una nube pública y sobre una privada.

- *Nube Virtual Privada (VPC):* este tipo utiliza la tecnología VPN por la que los proveedores de servicio pueden diseñar su propia topología y sus herramientas de seguridad como las reglas de los cortafuegos. Para muchas compañías esta tipología es la más adecuada porque proporciona una migración a un entorno de nube con impacto mínimo.

2.3. Implementaciones para la computación en la nube.

En estos párrafos se presentará un estado del arte de implementaciones para la computación en la nube.

2.3.1. Tecnologías de Computación en la Nube.

Resumen de las tecnologías utilizadas en los entornos de la nube.

2.3.1.1. Infraestructura de los centros de datos

Un centro de datos es un lugar donde se encuentran los dispositivos físicos y no es trivial el diseño y planificación de estos centros porque puede influir en la capacidad de procesamiento, la escalabilidad y el rendimiento de entornos de computación distribuidos [2].

Un diseño, ya probado, es el formado por tres capas, como se puede observar en la Figura 2. La primera es la capa de acceso donde se encuentran los racks de servidores físicamente conectados a conmutadores con un enlace de 1 Gbps. Estos conmutadores son conectados a otros 2 conmutadores para proporcionar redundancia con enlaces de 10 Gbps.

La capa de agregación provee de funciones como el servicio de dominio, balanceo de carga…. Y, la capa del núcleo proporciona conectividad a los conmutadores de la capa de agregación mediante unos enrutadores que aseguran un enrutamiento sin ningún punto de fallo.

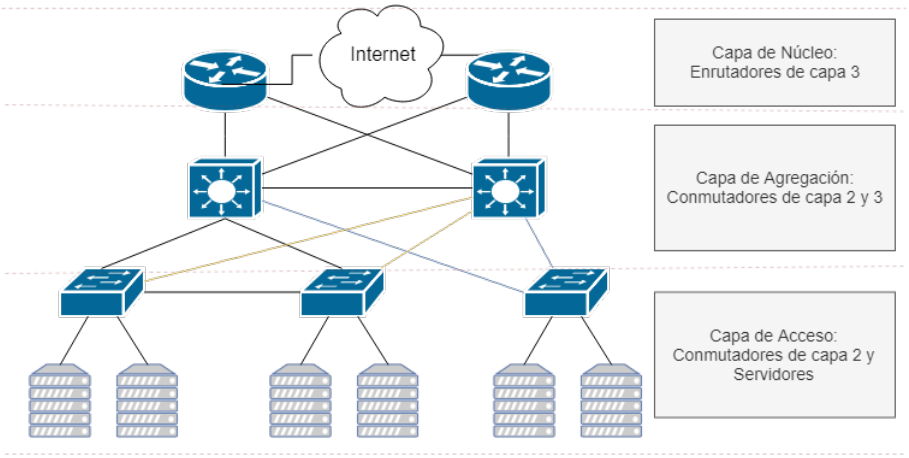


Fig. 2: Topología de un centro de datos

2.3.1.2. Sistema de archivos distribuidos.

Google File System (GFS) es un sistema distribuido desarrollado por *Google* y diseñado especialmente para proveer de unos datos de rendimiento muy altos, baja latencia y para sobrevivir a fallos de servidores individuales, además de, eficiencia, acceso fiable a los datos utilizando clústeres de servidores.

HDFS es el sistema de archivos de software libre creado por Apache que ofrece redundancia en el almacenamiento de los ficheros por medio de envío de bloques iguales entre varios nodos del sistema mediante la creación de bloques iguales de un fichero y almacenando éstos en nodos de almacenamiento diferente. Además, consta también de un sistema de replicación por lo que se pueden realizar copias de diferentes bloques en diferentes dispositivos proporcionando dos niveles de replicación (de fichero y de bloque) [4].

2.3.1.3. *Frameworks* distribuidos.

En los modernos centros de datos se encuentran servidores que utilizan *frameworks* para aplicaciones web como *Java EE*, pero también otros para la utilización de trabajos de computación intensiva como análisis financiero o animaciones de películas.

Un ejemplo es *MapReduce*, que es un software introducido por *Google* para computación distribuida de grandes conjuntos de datos en clústeres de ordenadores. Consiste en un nodo principal (nodo máster) a quién se envía trabajos *MapReduce*. Éste envía el trabajo los nodos de tareas disponibles dando prioridad, en caso de que no pudieran enviarse a uno, a nodos que estén en el mismo grupo de ordenadores minimizando el tráfico de red por el cable troncal evitando cuellos de botella y aumentando el rendimiento. En caso de que la tarea sea fallida o se agote se reprograma. Si el nodo máster falla todas las tareas se pierden, aunque el nodo máster tiene un registro en el sistema de archivos para que cuando éste sea reiniciado se vuelva a reiniciar el trabajo desde donde se dejó. Muy utilizado para el cálculo intensivo [3].

2.3.2. Productos comerciales más utilizados.

2.3.2.1. *Amazon Web Services.*

Conjunto de servicios en la nube que proporciona computación en la nube, almacenamiento y la posibilidad de desarrollar aplicaciones y servicios [5].

Amazon Elastic Compute Cloud (Amazon EC2) permite a los usuarios, mediante máquinas virtuales que utilizan el motor *Xen*, la configuración de instancias de servidores en los centros de datos utilizando *APIs* o herramientas específicas. El usuario después de crear y empezar una instancia puede subir software y provocar cambios. Cuando estos cambios han acabado pueden ser incluidos como una copia así una copia nueva puede ser lanzada cada vez.

Las imágenes de las máquinas virtuales de *EC2* son almacenadas y recuperadas de un *Amazon Simple Storage Service*. *Amazon S3* almacena los datos como objetos, de 1 byte a 5 gigabytes, que están agrupados en cubos y estos almacenados en regiones, que pueden ser elegidas para minimizar latencia, costes etc.

Amazon Virtual Private Cloud es seguro y la conexión para la migración de las infraestructuras TI de la empresa y la nube *AWS*, utilizando la conexión *VPN* para este fin además de proveer herramientas para la seguridad; reglas de cortafuegos, sistemas de detección de intrusión.

Amazon CloudWatch es una herramienta de monitorización en tiempo real de métricas sobre el *EC2*, como la utilización de la *CPU*, lectura/escritura en disco….

2.3.2.2. *Windows Azure.*

Conjunto de tres componentes que proveen de conjuntos especiales de servicios a los usuarios. *WA* provee de un entorno *Windows* para la prueba de aplicaciones y el almacenamiento de datos; *SQL Azure* proporciona servicios de datos basados en *SQL Server* y *.NET* ofrece servicios distribuidos de infraestructura entre aplicaciones locales y en la nube.

Soporta la creación de aplicaciones en *.NET Framework* como *C#*, *Visual Basic*, *C++*, además de la creación de aplicaciones web con *ASP.NET* y *WCF*, y permite almacenar datos mediante el acceso *REST* con *HTTP* o *HTTPS*.

2.3.2.3. *Google App Engine.*

Es una plataforma para aplicaciones web en centros administrados por *Google*. Soporta lenguajes como *Python* y *Java*, además de *Django*, *CherryPi* y escritos por *Google* similares a *JSP* o *ASP.NET*. *Google* se encarga de desplegar el código en un clúster con aquello necesario. Las versiones más actuales de *APIs* soportan el almacenamiento y recuperación de datos de una base de datos no-relacional mediante requerimientos *HTTP* [4].

2.4. Fundamentos teóricos.

En este apartado se recorrerán los distintos conceptos teóricos incluidos en este trabajo, así como la definición de éstos.

2.4.1. Arquitectura Cliente - Servidor.

En este apartado se detalla la arquitectura cliente-servidor cuya implementación será necesaria para el desarrollo de este proyecto.

2.4.1.1. Definición.

La arquitectura cliente-servidor es una arquitectura de computación en la que se consigue un procesamiento cooperativo mediante un conjunto de procesadores de tal forma que *n* clientes solicitan servicios de computación a *m* servidores.

Siguiendo esta visión descentralizada, la arquitectura cliente-servidor consiste en una arquitectura distribuida de computación en la que las tareas de cómputo se reparten entre distintos procesadores obteniendo el resultado final de forma transparente independientemente del número de servidores que han intervenido en el tratamiento [9].

2.4.2. Elementos principales.

2.4.1.2.1. Cliente.

Un cliente es el proceso que permite al usuario formular los requerimientos y enviarlos al servidor, también conocido como *front-end*.

La principal función del cliente es manejar las funciones relacionadas con la manipulación y despliegue de datos, por lo que éstos son desarrollados sobre plataformas que permiten construir interfaces de usuario, además de poder acceder a los servicios distribuidos en cualquier parte de la red.

Finalmente, la funcionalidad del proceso cliente marca la operativa de la aplicación. De este modo un cliente puede ser clasificado como:

- Cliente basado en aplicación de usuario: si los datos son de baja interacción y están fuertemente relacionados con la actividad de los usuarios de esos clientes.

- Cliente basado en lógica de negocio: toma datos suministrados por el usuario y efectúa los cálculos necesarios según los requerimientos del usuario.

2.4.1.2.2. Servidor.

Un servidor es todo proceso que proporciona un servicio a procesos cliente. Por esto, es el proceso encargado de atender a múltiples clientes qué hacen peticiones de algún recurso administrado por él. Es conocido como proceso *back-end* y normalmente maneja todas las funciones relacionadas con las reglas del negocio y los recursos de datos.

2.4.1.2.3. *Middleware*.

El *middleware* es un módulo intermedio que actúa como conductor entre sistemas permitiendo a cualquier usuario de sistemas de información comunicarse con varias fuentes de información que se encuentran conectadas por una red.

La utilización de éstos permite desarrollar aplicaciones con arquitecturas cliente-servidor independizando los servidores y clientes, facilitando la interrelación entre ellos y evitando dependencias de tecnologías propietarias.

El *middleware* se estructura en tres niveles:
1. Protocolo de transporte.
2. Sistema operativo de red *(N.O.S).*
3. Protocolo específico del servicio.

2.4.1.3. Sistema Operativo de Red *(N.O.S).*

Un sistema operativo de red es un software que permite la interconexión de ordenadores con la finalidad de acceder a los servicios y recursos mediante la creación de redes de computadores posibilitando la comunicación de un sistema informático con otros equipos utilizando una red. Incluye las principales funciones de comunicación de esta arquitectura.

2.4.1.3.1. Formas de comunicación:

2.4.1.3.1.1. Comunicaciones *Peer to Peer (P2P)*:

P2P o "*Peer to Peer*" permite la comunicación y compartición de información entre clientes sin necesidad de la figura de un servidor central que facilite la comunicación. Un ejemplo de este tipo de comunicaciones es el *Socket* por el cual dos clientes pueden intercambiar cualquier flujo de datos utilizando el *Protocolo TCP/IP* y qué queda definido por un conjunto de datos como son; las direcciones IP (tanto del cliente emisor como del cliente receptor), un par de números de puerto (tanto del cliente emisor como del cliente receptor) y un protocolo de transporte [10].

2.4.1.3.1.2. Colas de mensajes (MOM):

MOM o "Middleware Orientado a Mensajes" permite la comunicación mediante envío de mensajes a aplicaciones distribuidas de una forma asíncrona.

2.4.1.3.1.3. *Llamada a Procedimiento Remoto* (RPC):

"Llamada a Procedimiento Remoto" o *RPC* es la forma de comunicación mediante la que una máquina cliente ejecuta una llamada a un procedimiento o programa ubicado en una máquina diferente ofreciendo deslocalización puesto que las dos máquinas se localizan en distintos lugares, heterogeneidad, seguridad y transparencia de distribución [9].

2.4.1.4. Tipología de la arquitectura cliente - servidor.

Según las necesidades obtenidas en la etapa de análisis de un proyecto, es necesario conocer las distintas tipologías de esta arquitectura para conocer las restricciones de las diferentes tipologías para poder hacer las consideraciones y estimaciones de la configuración teniendo en cuenta; la oportunidad de información, tiempos de respuesta, tamaños de registros, tamaños de bases de datos, estimaciones del tráfico de red, etc. Existen dos tipologías de arquitectura cliente - servidor; según el tamaño de componentes y según la naturaleza del servicio proporcionado.

2.4.1.4.1. Según el tamaño de componentes.

Está tipología se basa en la libertad para balancear la carga de proceso entre los niveles de presentación, aplicación y base de datos dependiendo de qué segmento de las capas de software tengan que soportar la mayor o menor carga.

2.4.1.4.1.1. *Fat Client (Thin Server)*.

En este esquema el peso de la aplicación es ejecutada en el cliente (nivel de presentación y aplicación en un único proceso cliente) dejando al servidor las funciones que provee un administrador de base de datos. Este tipo de arquitectura tiene mejor aplicación en sistemas de apoyo a decisiones (*DSS*) y en sistemas de información ejecutiva (*EIS*).

2.4.1.4.1.2. *Fat Server (Thin Client).*

En este esquema el proceso cliente solamente muestra la interfaz de usuario mientras que el peso de la aplicación corre en el servidor de aplicación. Es una arquitectura que presenta mayor flexibilidad para el desarrollo de gran variedad de aplicaciones, incluyendo los sistemas de misión crítica a través de servidores de transacciones.

2.4.1.4.2. Según la naturaleza de servicios proporcionados.

2.4.1.4.2.1. Servidores de Ficheros.

Un cliente realiza un requerimiento de dichos ficheros sobre la red. Los servidores de ficheros utilizan recursos compartidos sobre la red y son necesarios para la creación de repositorios de documentos, imágenes y archivos de gran tamaño.

2.4.1.4.2.2. Servidores de Bases de Datos.

Permiten a un proceso cliente solicitar datos y servicios directamente a un servidor de bases de datos cuyo servidor deberá proveer un acceso compartido a los datos con los mecanismos de protección necesarios. concurrencia, seguridad y consistencia de los datos.

2.4.1.4.2.3. Servidores de Transacciones.

Permite al proceso cliente llamar a funciones, procedimientos o métodos que residen en el servidor, ya sea de un servidor de bases de datos como un servidor de aplicaciones.

En este tipo de arquitectura el intercambio a través de la red se realiza mediante un único mensaje de solicitud/respuesta independientemente de que se necesite ejecutar una o más funciones agrupadas en una unidad lógica denominada transacción evitando la solicitud/respuesta de un mensaje por cada sentencia *SQL*.

Estas aplicaciones denominadas *OLTP* ("*On Line Transaction Processing*") están orientadas a dar soporte a los procedimientos y reglas de los sistemas de misión crítica como por ejemplo los cajeros automáticos.

2.4.1.4.2.4. Servidores de Objetos.

Las aplicaciones cliente-servidor son escritas como un conjunto de objetos cliente que se comunican con los objetos servidores utilizando un *Object Request Broker* (*ORB*). El cliente invoca un método de un objeto remoto, el ORB localiza el método del objeto en el servidor y lo ejecuta para devolver el resultado al objeto cliente [9].

2.4.1.5. Modelos de cliente-servidor.

Clasificación de las arquitecturas cliente-servidor basada en la idea de planos (*tier*). Se trata de definir el modo en que las prestaciones funcionales de la aplicación serán asignadas, y en qué proporción. Dichas prestaciones se agrupan en tres componentes, que coinciden con cada plano; interfaz de usuario, lógica de negocios y datos compartidos.

2.4.1.5.1. Modelo cliente-servidor de 2 capas.

Este modelo se caracteriza por la conexión directa entre el proceso cliente y un administrador de bases de datos conectados vía *LAN* a un servidor de aplicaciones local otorgando el acceso a los datos administrados por él mismo.

2.4.1.5.2. Modelo cliente-servidor de 3 capas.

Este modelo se caracteriza por la elaboración de la aplicación en base a dos capas de software más la capa correspondiente del servidor de bases de datos pudiéndose balancear la carga de trabajo entre el proceso cliente y el nuevo proceso de servidor de aplicación. En este modelo

el servidor local tiene un comportamiento dual puesto que actúa como cliente o servidor en función de la dirección de la comunicación.

2.4.2. *Code Offloading Methods*.

"*Code Offloading*" o externalización de código es el paradigma de la computación que se ocupa de resolver la ejecución de fragmentos de software, que requieren un alto coste de procesamiento, en un servidor en la nube desde máquinas con recursos limitados para acelerar el proceso de ejecución de este fragmento software y reducir el consumo de energía en dispositivos de batería limitada.

Este paradigma, descrito de forma superficial en la Figura 3, aprovecha de un modelo general por el cual los códigos de computación intensiva de una aplicación se evalúan mediante un proceso de toma de decisiones basado en la disminución del consumo de energía o el aumento de capacidad de procesamiento para determinar si se descarga en la nube o no. A continuación, el fragmento de código se descarga en los recursos de la nube remotamente.

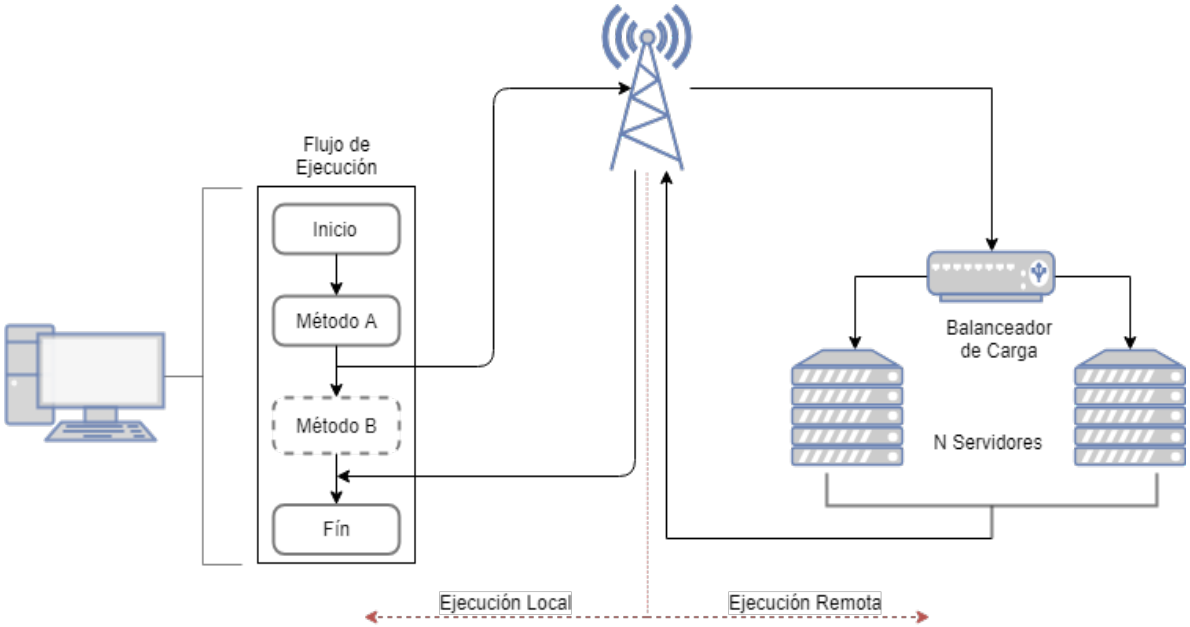


Fig. 3: Proceso de Externalización de Código a la Nube

Por tanto, este paradigma permite a los dispositivos migrar parte del cómputo desde dispositivos con recursos limitados a máquinas de computación con mayores recursos.

Una desventaja de este paradigma es la identificación y partición del código a descargar en la nube por parte de los desarrolladores enfocando esta tarea en cuatro categorías: descarga particionada, migración de máquinas virtuales, descarga basada en agentes móviles y la descarga basada en replicación.

2.4.2.1. Descarga particionada.

Los métodos englobados en esta categoría únicamente la parte de computación intensiva de las aplicaciones se identifica y descarga a los servidores remotos. También, se introduce el modelo de computación Cliente - Servidor convencional para realizar la descarga particionada.
Bajo el modelo de computación Cliente - Servidor, los dispositivos con recursos limitados se consideran clientes ligeros, "*thin client*", mientras que los recursos informáticos remotos que ejecutan los códigos descargador se consideran servidores gordos o "*fat servers*".

La aplicación puede ser particionada de manera estática, antes de la ejecución, o de manera dinámica, durante la ejecución. En la partición estática, los fragmentos de código a descargar están decididos y programados en la aplicación por los desarrolladores. De esta manera, no hay una sobrecarga impuesta por la decisión de particiones cuando la aplicación se está ejecutando, pero requiere el conocimiento completo del programa de ejecución ofreciendo un enfoque menos adaptable a un entorno informático dinámico siendo un enfoque poco utilizado en tareas ejecutadas en la nube. Como solución se incorpora otro enfoque llamado partición dinámica, que al contrario que la estática, analiza los procesos de una aplicación en tiempo de ejecución con la ayuda de la lógica de decisión. Las técnicas de partición dinámica más utilizada son la programación basada en flujo, la programación en tiempo de ejecución en .*Net*, *Java Reflection* y la memoria compartida distribuida [11].

2.4.2.1.1. Programación basada en Flujo.

La programación basada en flujo utiliza un paradigma de programación de flujo de datos para modelar aplicaciones como la interconexión de distintos procesos modulizados mediante conexiones comunicación predefinidas interconectando distintos módulos entre sí de formas diferentes para lograr distintos programas sin tener que modificar nada. Para aplicaciones móviles soportadas por el sistema operativo Android, fue introducido *JavaFBP* sin embargo requiere la instalación tanto en dispositivos cliente como en servidores la instalación de esta herramienta, lo que dificulta la escalabilidad de las aplicaciones. Como solución se aportó *Java Reflection*.

2.4.2.1.2. "*Java Reflection*".

Programa proporcionado por Java que permite inspeccionar y manipular clases, interfaces, campos y métodos anotados en tiempo de ejecución pudiendo así decidir dinámicamente, en tiempo de ejecución, que descargar y ejecutar cualquier partición sin requisitos de instalaciones adicionales. *ThinkAir* fue presentado para la descarga de código del cloud utilizando *Java Reflection*. En esta propuesta los métodos anotados se evalúan mediante la lógica de decisión y se descargan en máquinas virtuales en la nube que ejecutan clones de *Android*. Con esta herramienta se pueden ampliar o reducir dinámicamente el número de máquinas virtuales en la nube, aunque solamente se puede descargar un fragmento a la vez pudiendo llegar a causar problemas de bloqueo ("*lock-in"*).

2.4.2.1.3. Memoria Compartida Distribuida.

"*COMET*" es una solución al problema de bloqueo dado en el punto anterior para Android utilizando la memoria compartida distribuida. Está solución aprovecha el modelo de memoria de Java y la sincronización de la máquina virtual para crear una memoria compartida distribuida para la migración de código entre máquinas.

La descarga se implementa al sincronizar la información del heap, pilas de memoria, estados de registro y clases sintéticas entre máquinas virtuales cloud y dispositivos.

2.4.2.1.4. Programación en tiempo de ejecución.

"*.NET Common Language Runtime*" (CLR) proporciona una plataforma de codificación que ofrece integración entre lenguaje, manejo de excepciones y un modelo simplificado para la interacción de componentes. Un ejemplo de aplicación de CLR es *MAUI*. En *MAUI* los métodos anotados por las etiquetas de CLR se evalúan mediante la lógica de decisiones en tiempo de ejecución para decidir si se van a descargar. Este marco obliga a desarrollar una aplicación para el servidor cloud perjudicando así la escalabilidad de la aplicación

2.4.2.2. Migración de Máquinas Virtuales.

Este enfoque se basa en la idea de mover el cálculo de forma dinámica entre máquinas sin interrumpir la ejecución en curso. Es un enfoque adecuado para el aumento de potencia de cómputo en el cloud ya que la mayoría de las aplicaciones móviles se ejecutan en máquinas virtuales pudiéndose migrar dichas máquinas de una forma parcial o completa según el requisito de la descarga.

Este enfoque tiene un inconveniente puesto que la migración de una imagen de máquina virtual y su estado puede dar una sobrecarga puesto que se transfiere mediante una red inalámbrica llegando a ser costoso e ineficiente en condiciones inestables de la red. Por esto la finalidad de este enfoque es mejorar la eficiencia energética considerando las condiciones de la red.

"*Cloudlet2*" es un marco propuesto basado en la migración de máquinas virtuales para llevar el cálculo del dispositivo móvil a los servidores cercanos para superar la larga latencia causada por la transferencia de datos al cloud a través de redes inalámbricas. *Cloudlet* se refiere a un conjunto de computadoras rica en recursos bien conectada a Internet y disponible para su uso en dispositivos cercanos.

"*CloneCloud*" es un marco de externalización de código que funciona con *DalvikVM* y *Microsoft .Net* que despliega uno o más clones de la aplicación y sus datos en las máquinas virtuales de la nube y servidores cercanos utilizando la intercepción de procesos para ejecutar partes de la aplicación en la nube. Los estados tanto de ejecución y de la pila son sincronizados entre los dos procesos, del dispositivo y la máquina virtual en el cloud. Aunque

ofrece limitación en las funciones a descargar debido a la imposibilidad de acceder a todos los datos nativos en el lado de la nube [11].

2.4.2.3. Migración basada en Agentes Móviles.

Un agente móvil es un software que se puede transferir desde un dispositivo móvil a una red y recorrerá todas las máquinas de la red. Cuando una aplicación utiliza un agente móvil necesita solicitar un servicio desde el servidor remoto, recopila la información de ejecución de la aplicación y pasa al entorno de ejecución del agente para la descarga y ejecución remota.

La utilización de agentes móviles tiene múltiples ventajas como la comunicación asíncrona, posibilidad de trabajar en el dispositivo mientras que se está ejecutando la parte en la nube. Son robustos a fallos de servidor puesto que toda la información necesaria es transferible al servidor. Un ejemplo de agente móvil es *JADE* ("*Java Agent Development Environment*") aunque este agente no considera la *QoS* de los recursos en la nube como el balanceo de carga, "*multi-tenancy*" y la eficiencia energética del canal inalámbrico para múltiples usuarios.

2.4.2.4. Descarga basada en Replicación.

Debido a los inconvenientes expresados en las anteriores técnicas M.S. Gordon [10] ha propuesto un enfoque diferente mediante el uso de replicaciones. Propuso "*Tango*" que intenta reducir la latencia de la aplicación por parte del usuario al cambiar la ejecución y la visualización de salida automáticamente entre la aplicación y su repica en máquinas remotas para la más rápida. "*Tango*" tiene limitaciones puesto que solamente una réplica puede liderar la ejecución en lugar de utilizar trabajo de simultáneo [11].

2.4.3. GPU Computing.

La cantidad de núcleos ofrecidos en la tarjeta gráfica o *GPU* permite que las aplicaciones descarguen porciones de código que necesiten un uso intensivo de cómputo a la *GPU* mientras que las resultantes se ejecuten en la *CPU* [12, 13].

"*MOCHA"* es una arquitectura basada en el Cloud que utiliza la GPU para obtener un mayor rendimiento en aplicaciones de reconocimiento de objetos. El "*Cloudlet*" en "*MOCHA"* se utiliza para reducir la latencia al permitir que dispositivos se conecten a "*Cloudlet"* mediante enlaces de baja latencia y posteriormente "*Cloudlet"* procesa parte del cálculo intensivo.

Las *GPUs* se utilizan para realizar el procesamiento masivo en paralelo mediante el uso de *CUDA11*, plataforma de programación en lenguaje C, con librerías para Java, y modelos de programación diseñados para la programación en paralelo de la *GPU.*

2.4.4. Conclusiones

Una vez expuesta las distintas fórmulas para la realización de una externalización de código para su procesamiento, en los sucesivos experimentos se utilizará una arquitectura Cliente – Servidor mediante Sockets TCP puesto que es la forma de comunicación Cliente – Servidor más extendida además de aquellas ventajas que aporta este método como la sencilla forma de comunicación utilizando únicamente la dirección IP tanto de Cliente como de Servidor y los puertos para realizar la conexión.

Tras la configuración de la comunicación, se ha creado un Modelo Cliente-Servidor de 2 capas donde el ordenador caracterizado como Cliente es un ordenador denominado *thin-client* y el ordenador caracterizado como Servidor es un ordenador denominado *fat-client* donde se implementará el código necesario para ejecutar la computación GPU para la realización del experimento que detallaremos en sucesivos puntos.

3. Arquitectura propuesta.

Como finalidad de este trabajo, además de ser explicado en la primera parte de éste, es la propuesta de arquitectura distribuida para la ejecución de un programa con funciones de alto requerimiento computacional de forma distribuida.

Por tanto, se propone una arquitectura de computación basada en la arquitectura cliente - servidor clásica remodelada para hacer frente a las exigencias explicadas en el párrafo anterior y que tienen como modelo la imagen contenida en la Figura 4 [19, 20].

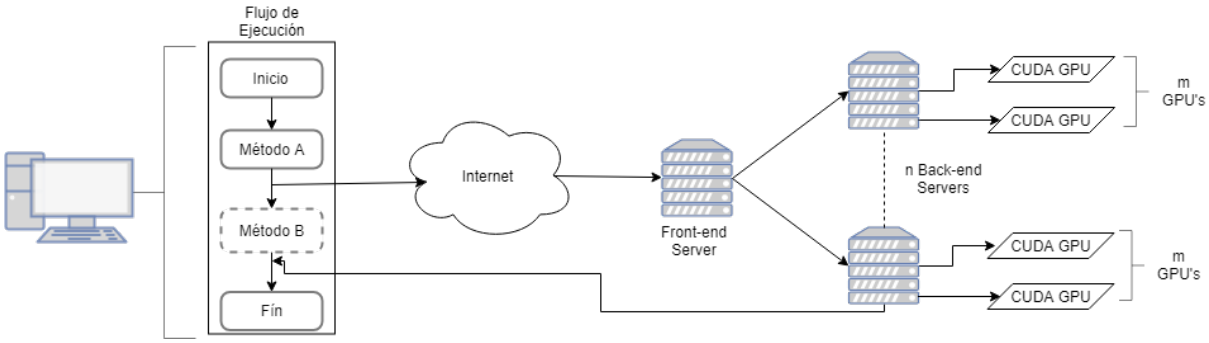


Fig. 4: Arquitectura distribuida para el procesamiento específico de operaciones *CAD/CAM.*

El funcionamiento de la arquitectura propuesta funcionará de la siguiente forma: El ordenador personal o *thin-client* ejecutará un software ligero que solamente realizará en la CPU o GPU local las funciones de bajo coste computacional y éste externalizará a un servicio en la nube aquellas funciones que el ordenador personal calcule de mayor coste computacional mediante una técnica llamada *Code Offloading* por el que se enviará a través de Internet una petición de cálculo a un servidor llamado *front-end* donde será recibida la petición, calculado el coste computacional y asignada a un servidor llamado *back-end* donde se ejecutarán los cálculos oportunos y serán devueltos al ordenador cliente para que puedan ser utilizados en el programa que el usuario utiliza [16, 17, 18].

Este servidor *back-end* cuenta con una ventaja y es la configuración según los requerimientos del *hardware,* así los usuarios de este servicio podrán disponer del *hardware* que mejor se adapte a las necesidades. Esta arquitectura utilizará la aceleración y computación específica aportada por las *GPU* mediante el lenguaje *CUDA* y podrán ser conectadas tantas tarjetas *GPU* como sean necesarias en los servidores de *back-end* ofreciendo mayor potencia de cálculo en casos en los que sea necesario y aportando un infinito número de combinaciones a los usuarios finales según sean sus necesidades.

3.1. Modelo de operación propuesto.

Tras la definición de la arquitectura en el punto 3, se utilizará un modelo de operación propuesto y que mostraremos en la Figura 5 y que también se utilizará como arquitectura para realizar los experimentos para los que se detallarán en el punto a tal efecto.

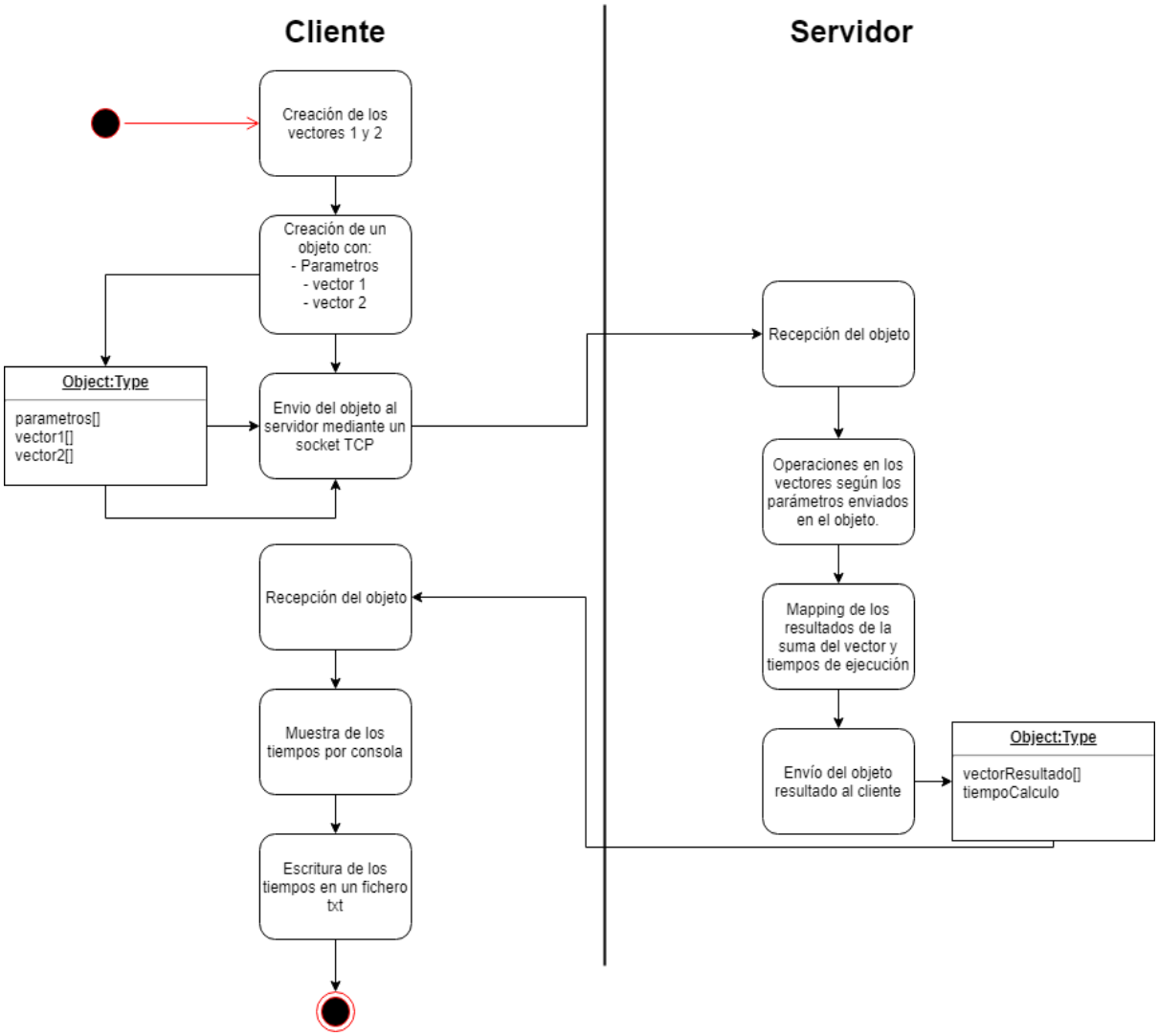


Fig. 5: Diagrama de flujo propuesto para la experimentación.

4. Experimentación.

A continuación, se presentarán distintos experimentos con los que se pretende investigar la idoneidad del paradigma de computación en la nube para implementar nuevas soluciones para el *software CAD/CAM*.

Para ello, se ha añadido un anexo a este trabajo donde se especifica el *hardware* de los equipos utilizados en las pruebas que se detallarán y con el fin de obtener resultados concluyentes y rigurosos, las distintas pruebas serán ejecutadas en el mismo *hardware* detallado en el anexo.

Para estos experimentos se ha desarrollado un software en lenguaje Java que proporciona servicios de conexión, utilizando Sockets TCP y aplicando la Arquitectura Cliente-Servidor explicada anteriormente, además de servicios para la operación matemática de suma de vectores tanto para calculo en el procesador (CPU) y el cálculo en la tarjeta gráfica (GPU) mediante CUDA (con librerías para Java, jCuda), cuya finalidad principal será la de medición temporal del cálculo de esta operación matemática como forma de comparar ambos modos de operar con grandes volúmenes de datos.

Para llevar a cabo la arquitectura propuesta y poder experimentar para presentar unas conclusiones verídicas se ha diseñado un diagrama de flujo donde se especifican los pasos y etapas por las que la suma de vectores que se llevará a cabo tendrá que pasar, ubicado en el punto 3.1, y por dar una secuencia lógica a la exposición de los experimentos y sus resultados, se expondrán a continuación de agrupados en tres bloques: *Pruebas de viabilidad de la propuesta*, *Pruebas de rendimiento* y *Pruebas de utilización del hardware.*

Los experimentos para apoyar este trabajo han seguido una línea lógica que empieza por los experimentos diseñados a probar y determinar la viabilidad de la propuesta creando para ello la arquitectura para probarlo correctamente. A continuación, se realizarán experimentos para determinar el rendimiento que ha sido llevado a cabo en el primer experimento para poder universalizar las medidas obtenidas para poder utilizar una cifra para comparar distintos paradigmas sin tener en cuenta variables externas que pueden llegar a ser determinantes. Finalmente, se ha probado en un entorno real, en este caso en las instalaciones de la Universidad de Alicante, la arquitectura propuesta con un numero variable de ordenadores simulando las peticiones de clientes, con la finalidad de probar la viabilidad de la propuesta en un entorno sin control.

4.1. Pruebas de viabilidad de la propuesta.

4.1.1. Experimento 1: Comparación temporal CPU – GPU de forma local.

En este primer experimento, se va a ejecutar un algoritmo simple como es la suma de vectores de la misma dimensión, que se aumentarán progresivamente, y con los mismos valores en su interior fijos.

Para este experimento se han declarado dos vectores de nombres A y B que serán los sumandos y otro vector C que será el que almacene los valores resultado de la suma de las componentes de A con las componentes de B. Por tanto, la finalidad de este experimento será la medición del tiempo de ejecución, de un programa en lenguaje C, utilizado en la suma de los vectores A y B, y que, mediante la comparación de los tiempos medidos, en el mismo dispositivo, con la ejecución del código directamente en la *CPU* y en el procesador gráfico (*GPU*) de una tarjeta gráfica utilizando el lenguaje de propósito específico llamado *CUDA*.

Los vectores de este experimento se han declarado de forma:

```
float A = new float[VECTOR_ELEMENTS]
float B = new float[VECTOR_ELEMENTS]
float C = new float[VECTOR_ELEMENTS]
```

y, se han añadido valores a A y B para que estos se interpreten como sumandos de la operación matemática que queremos realizar:

```
for (int i = 0; i < VECTOR_ELEMENTS; i++) {
        A[i] = aleatorio();
        B[i] = aleatorio();
},
```

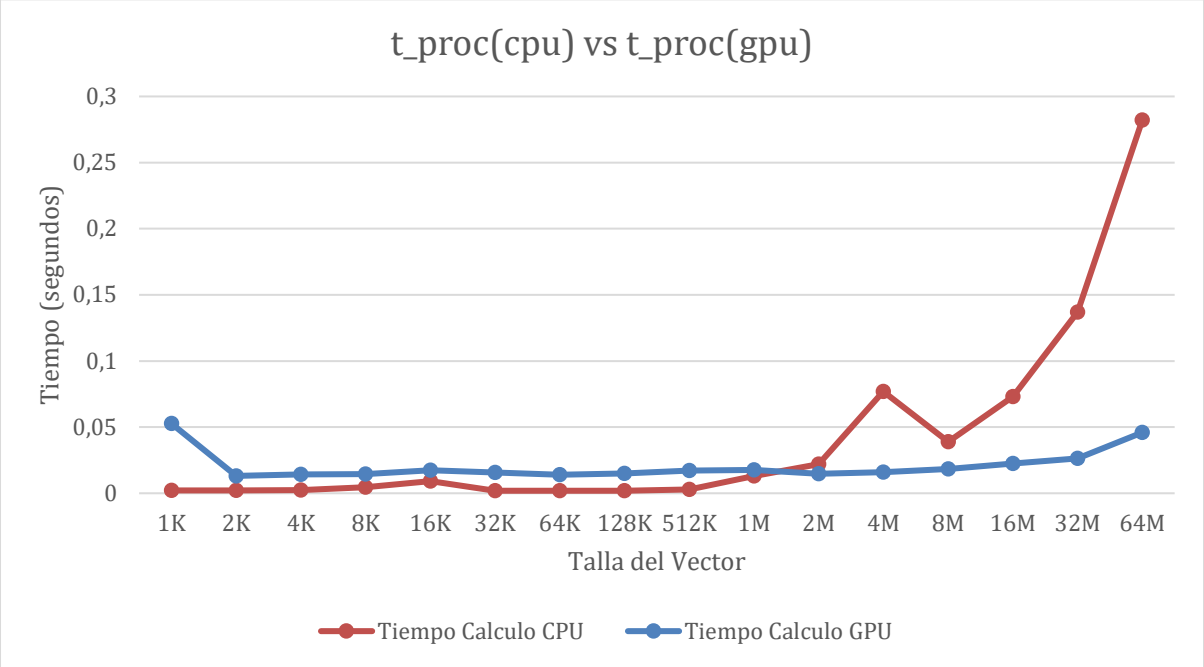
y se vuelquen una vez sumados al vector C mientras se realiza la medición de la operación suma, quedando así:

```
start_time = getTime();
for (j = 0; j < VECTOR_ELEMENTS; j++) {
        C[j] = A[j] + B[j];
}
end_time = getTime();
```

ofreciendo un tiempo de ejecución de la operación suma que será la igual a:

total_time = end_time – start_time.

Una vez realizado el código se han hecho pruebas, midiendo el tiempo de ejecución de los vectores cambiando la talla de estos, *VECTOR_ELEMENTS*, desde 2^{10} hasta 2^{25} elementos del vector con incrementos de una unidad en el exponente, tanto para la ejecución en la *CPU* del ordenador personal como para la ejecución del mismo código adaptado al leguaje *CUDA*.



Gráfica 1: Tiempo de procesamiento

Se ha realizado una gráfica que permite ver los resultados y analizarlos mediante la función de la línea de tendencia de los resultados.

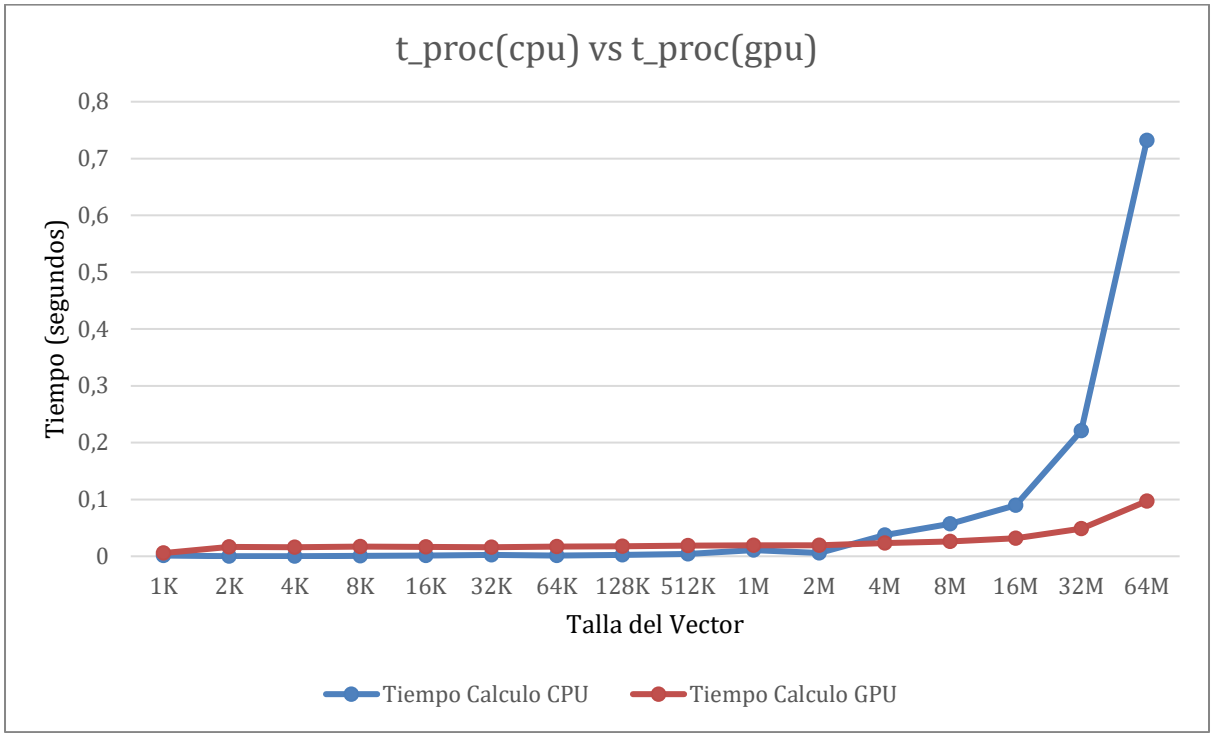
Tras el análisis de la gráfica y mediante la utilidad de la línea de tendencia, se puede observar cómo tras cálculos de mayor tamaño de los mostrados en esta gráfica, el tiempo empleado por la CPU será significativamente más elevado debido a la curva ascendente de la línea de tendencia.

4.1.2. Experimento 2: Comparación temporal CPU – GPU de forma remota.

Este experimento se compone a su vez de una serie de experimentos de misma naturaleza como el Experimento 1 pero utilizando un entorno de red distribuido intentando simular un entorno real. Por esto en este experimento se probarán conexiones cliente-servidor mediante comunicación Wireless, comunicación por LAN, comunicación por 3G y comunicación por 4G.

Para este experimento se utilizará el programa del experimento anterior únicamente modificado con servicios de envío y recepción de mensajes. Para esto se ha creado una clase Cliente donde se crearán dos vectores de tallas (2^{10} hasta 2^{25} elementos) cuyo interior sea un numero aleatorio y este enviará los vectores para la suma a una clase Servidor quien utilizará el cálculo tanto en la CPU como en la GPU y devolverá al cliente el vector resultado junto al tiempo utilizado tanto en el envío y recepción de los datos como el tiempo utilizado en calcular el vector resultado.

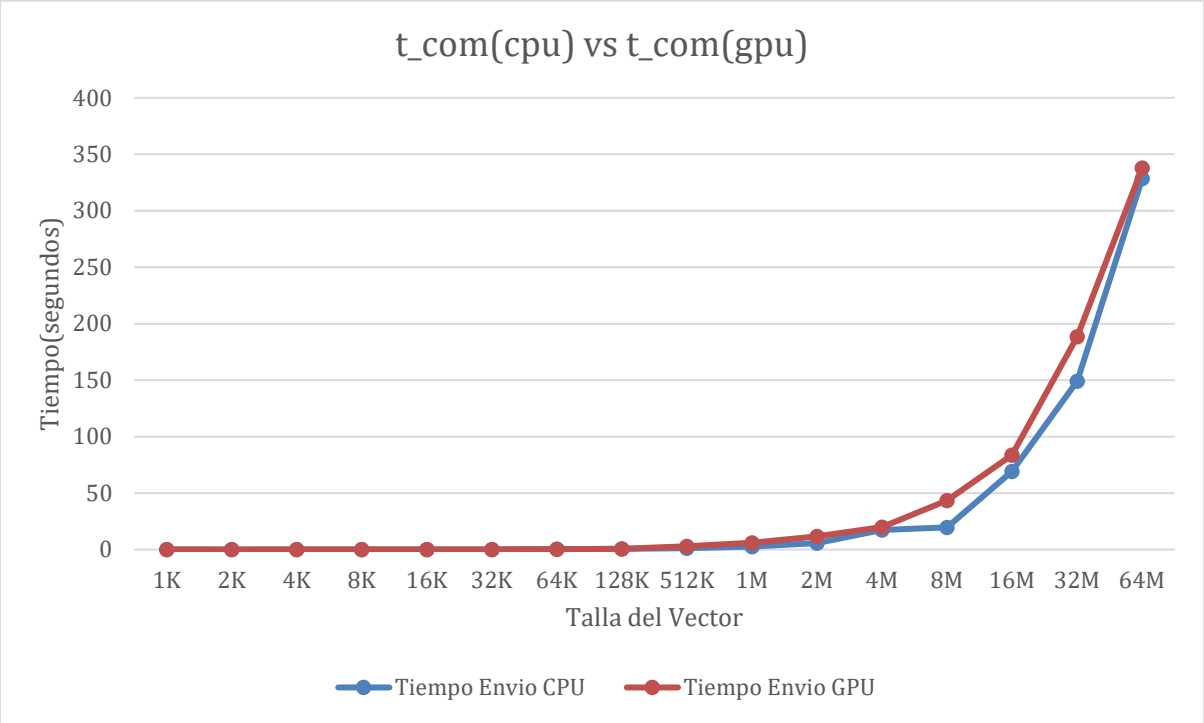
4.1.2.1. Experimento 2.1: Comparación temporal CPU-GPU de forma remota utilizando conexión Wireless.



Gráfica 2.1.1: Tiempo de procesamiento

Una vez explicado el código utilizado solamente queda añadir la gráfica con los resultados y la línea de tendencia de ambas formas de calcular el vector resultado.

También se ha creado una gráfica con el tiempo de envío y recepción calculada como tiempo de comunicación:

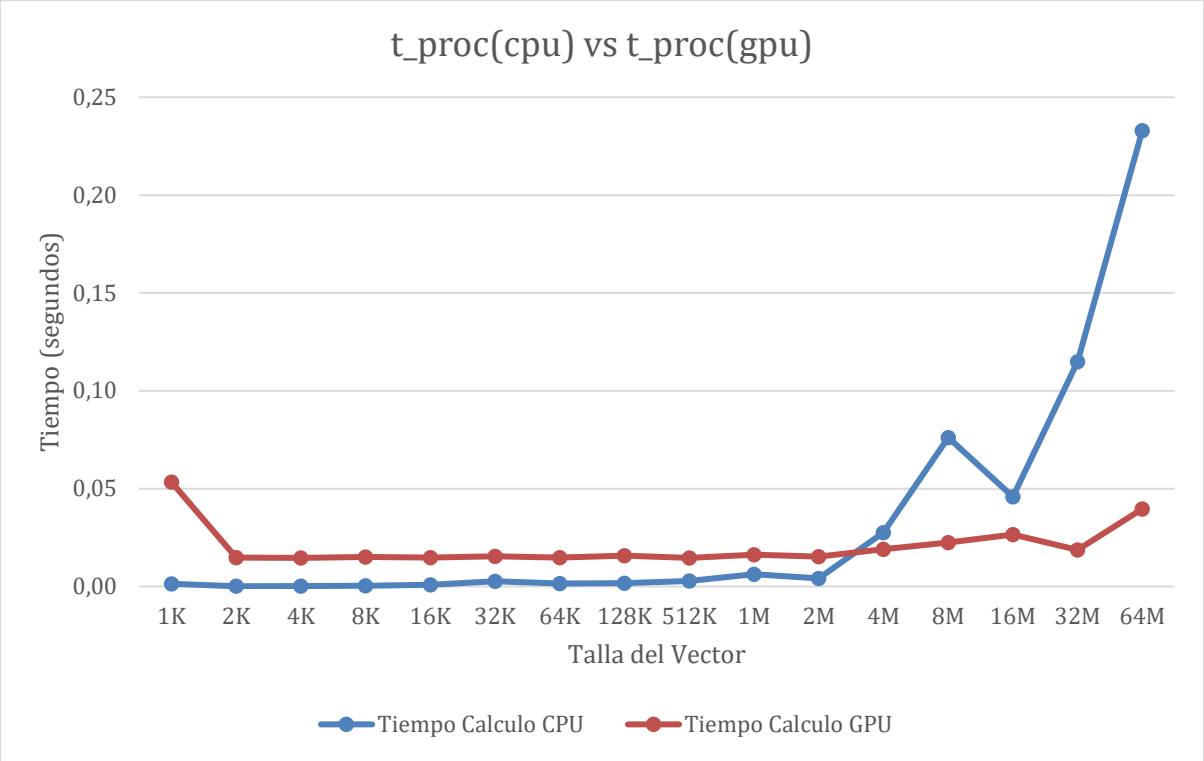


Gráfica 2.1.2: Tiempo de Comunicación.

Tras observar ambas gráficas se puede observar como el tiempo de procesamiento de cálculos aumenta en los cálculos de la CPU cuanto mayor es la talla del vector comparada con el tiempo de los cálculos de la GPU con CUDA. También aumenta el tiempo de envío de los vectores al servidor debido al aumento del tamaño de los vectores a través de la red.

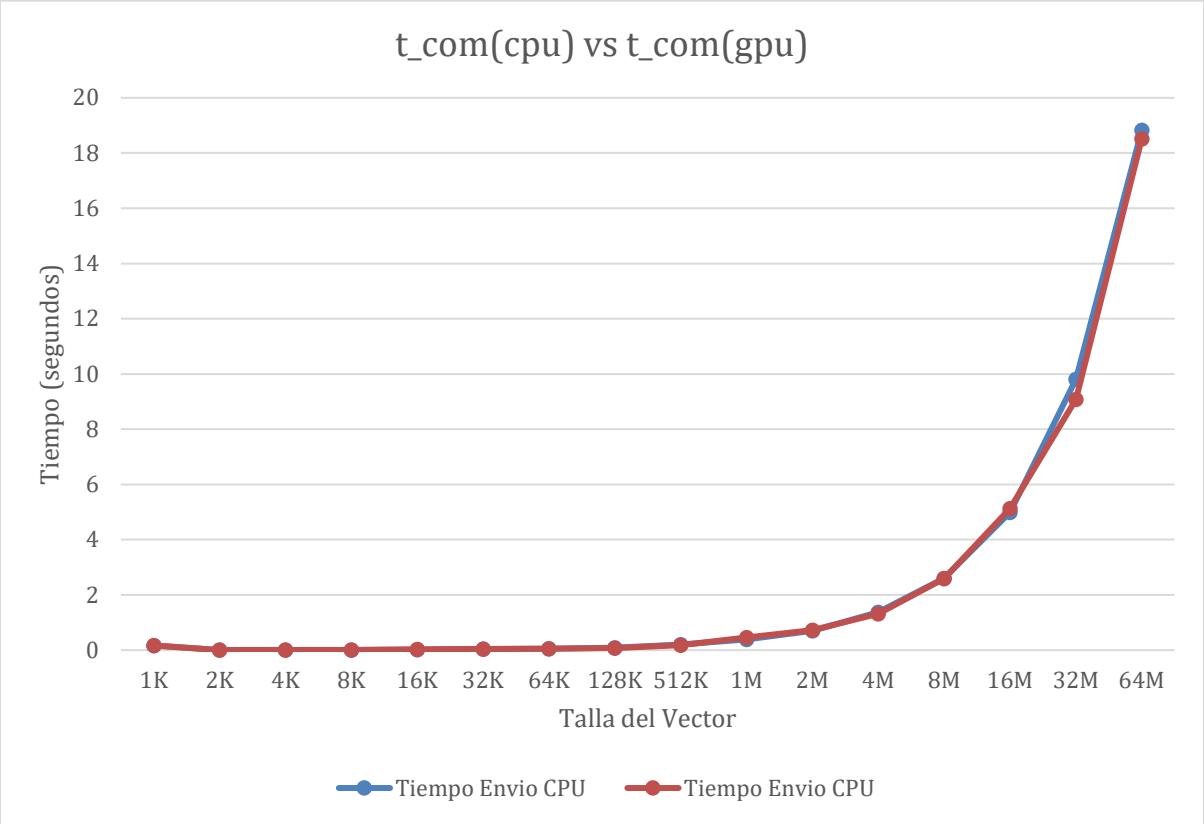
4.1.2.2. Experimento 2.2: Comparación temporal CPU-GPU de forma remota utilizando conexión LAN.

Una vez explicado el código utilizado solamente queda añadir la gráfica con los resultados y la línea de tendencia de ambas formas de calcular el vector resultado.



Gráfica 2.2.1: Tiempos de procesamiento

También se ha creado una gráfica con el tiempo de envío y recepción calculada como tiempo de comunicación:

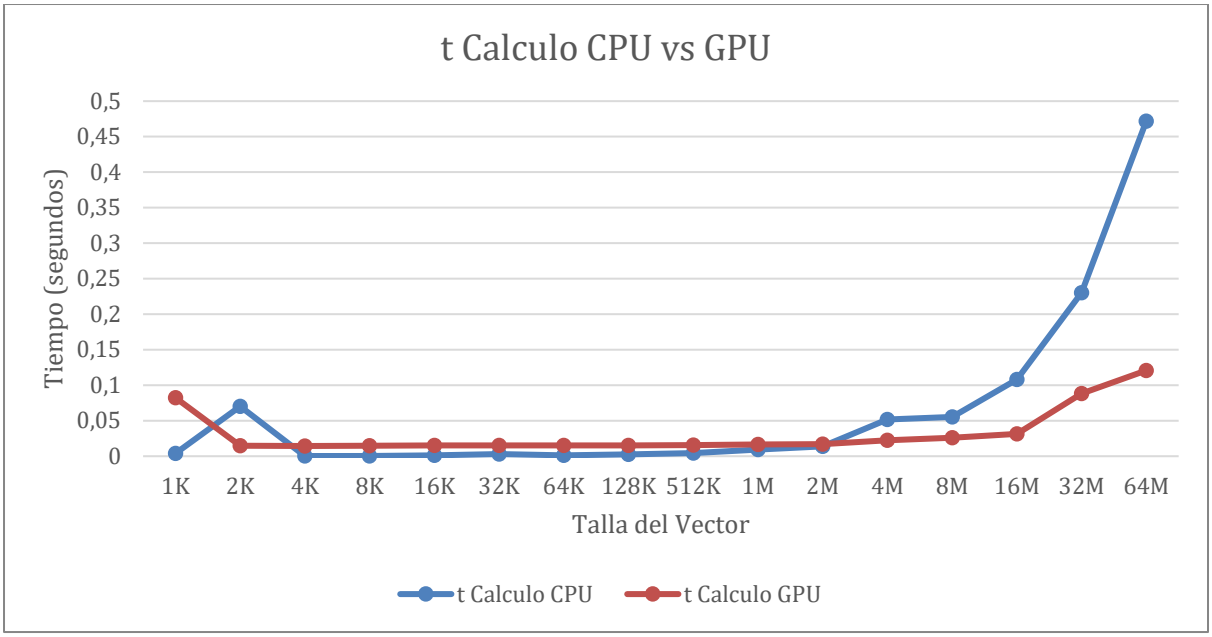


Gráfica 2.2.2: Tiempos de Comunicación

Tras observar las gráficas se puede ver como la línea de tendencia indica que el tiempo de procesamiento en la CPU aumenta de forma significativa mientras que para la GPU aumenta con intervalos menores. Además, también se puede observar en la gráfica 2 que el tiempo de comunicación aumenta con la talla del vector, aunque de forma similar, casi idéntica, debido al medio utilizado que por el contrario al anterior experimento que sufre de interacciones con medios físicos por la naturaleza de la comunicación Wireless.

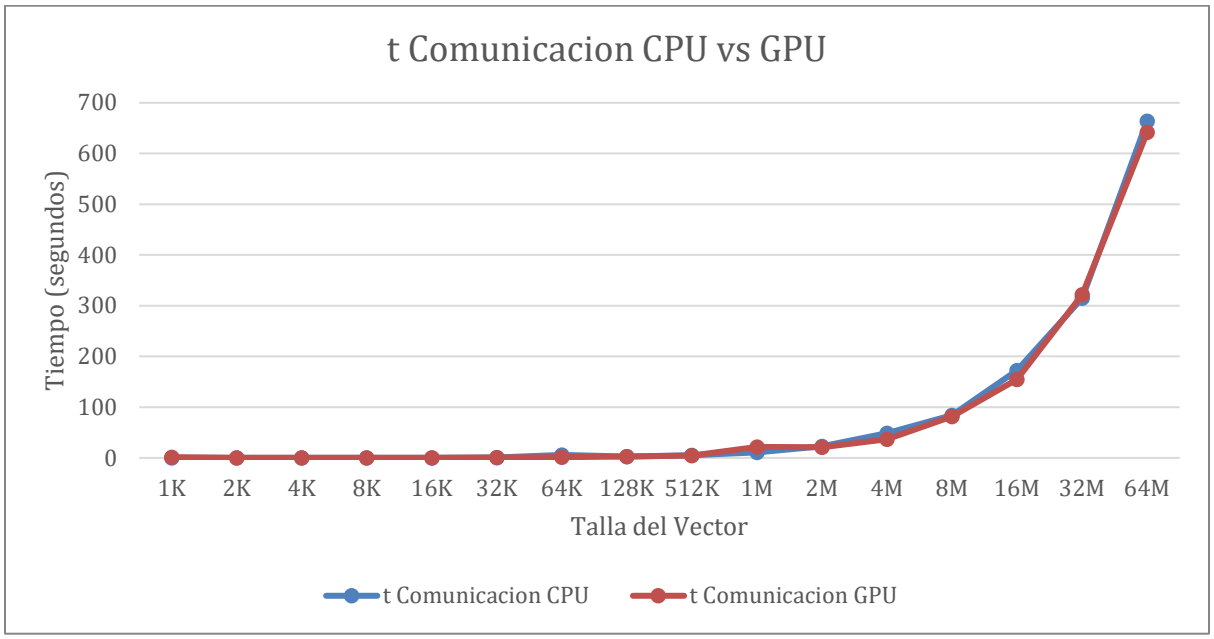
4.1.2.3. Experimento 2.3: Comparación temporal CPU-GPU de forma remota utilizando conexión 3G.

Para este experimento, se ha utilizado el código arriba explicado y cuya principal motivación es la utilización de la red 3G para la transferencia de los datos tanto al cliente como al servidor dando como resultado dos gráficas, una con los resultados del tiempo de calculo que ha empleado el Servidor y otra con los resultados de los tiempos de envío y recepción llamados tiempos de comunicación.
Este experimento se ha realizado 3 veces en 3 días diferentes en franjas de Mañana de 10:00 h a 12:00h, Tarde de 16:00 h – 18:00 h y Noche de 22:00 h – 00:00 h, con la finalidad de eliminar el factor carga de la red inalámbrica encontrado en algunas horas del día. Se ha llevado a cabo con unos porcentajes de cobertura del 60% al 65% por lo que la reproducción de este experimento en localizaciones distintas, así como con porcentajes de cobertura diferentes podría dar lugar a otros resultados difiriendo de los expuestos en este trabajo. Además, las gráficas mostradas corresponden con los datos obtenidos de la realización del promedio de los 3 resultados obtenidos (Mañana, Tarde y Noche) para el procesamiento en la CPU del Servidor y de los 3 resultados obtenidos (Mañana, Tarde y Noche) para el procesamiento en la GPU del Servidor.
Por esto, se incluye la gráfica con los datos de tiempo de cálculo:



Gráfica 2.3.1: Tiempos de Procesamiento

Y, la gráfica con los tiempos de comunicación Cliente – Servidor:



Gráfica 2.3.2: Tiempos de Comunicación.

Como una de las finalidades de este experimento ha sido lograr que no se vea afectado el experimento por la sobrecarga de la red y por este motivo el experimento se ha realizado en 3 franjas, se ha recogido los tiempos de las distintas ejecuciones y se ha configurado una gráfica con los datos de comunicación de las 6 ejecuciones (3 para la ejecución CPU y 3 para la ejecución GPU)

4.1.3. Experimento 3: Comparación temporal CPU-GPU en una arquitectura Cloud simulada.

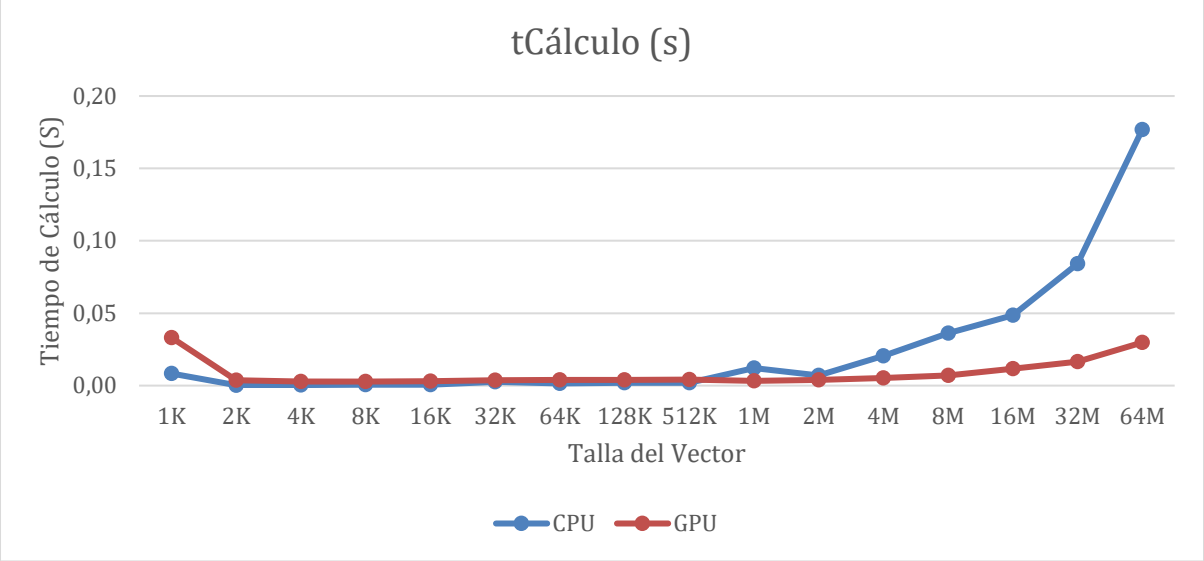
En este experimento se han utilizado dos ordenadores deslocalizados que ejecutan las funciones de cliente y servidor con la finalidad de recrear el experimento número 2 utilizando como servidor un ordenador equipado adecuadamente y situado en el laboratorio del Departamento de Tecnologías de la Información y Comunicación de la Universidad de Alicante, a modo de servidor Cloud para analizar la viabilidad de la externalización de código de alto requerimiento de hardware a un servidor externo.

Para esto se ha necesitado únicamente una IP pública para la conexión del socket cliente al servidor Cloud y proporcionada por el servicio técnico del mismo laboratorio además de proporcionar todo lo necesario como la excepción en el Firewall, la redirección de un puerto público a uno privado y la posibilidad de control remoto de la máquina servidor.

Finalmente, solamente detallar que los experimentos que se exponen a continuación han sido realizados mediante las mismas técnicas y finalidades de los experimentos del "*apartado 5.2.*" de este mismo trabajo.

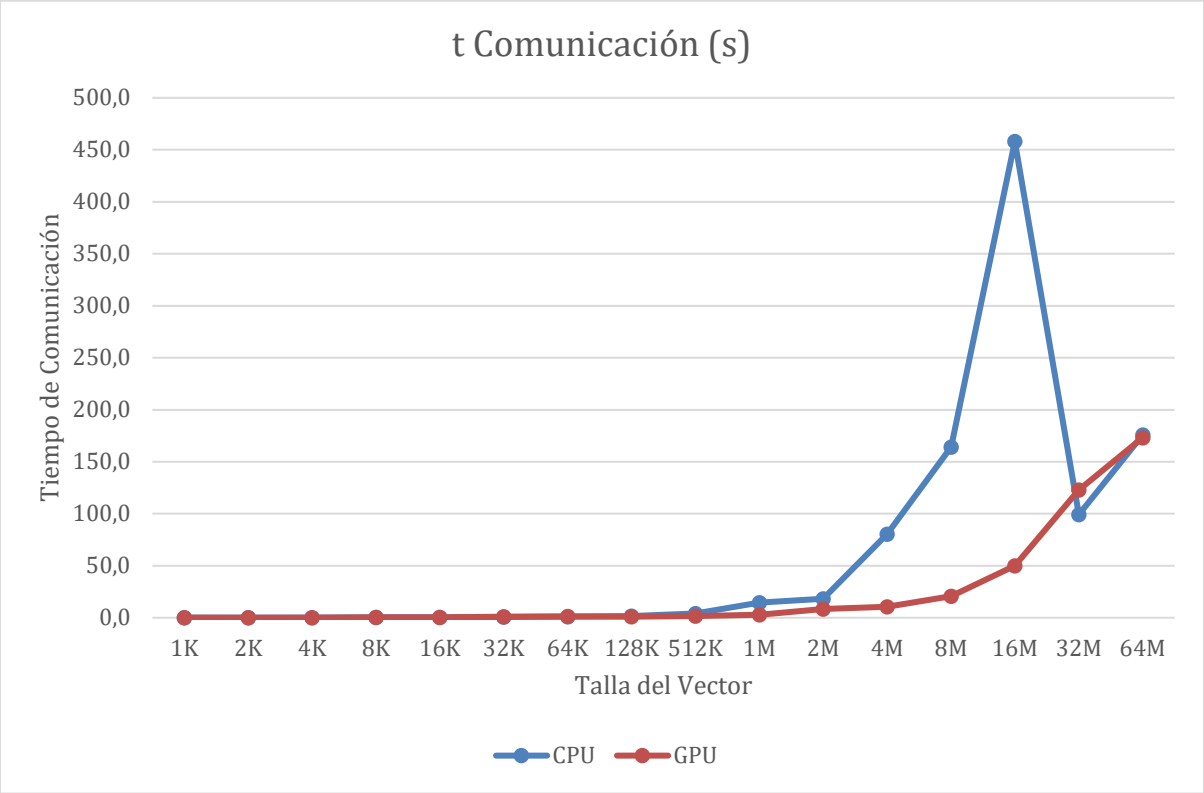
4.1.3.1. Experimento 3.1: Comparación temporal CPU-GPU de forma remota al servidor Cloud utilizando conexión Wireless.

Se ha obtenido una tabla de resultados de tiempo de cálculo:



Gráfica 3.1.1: Tiempos de Procesamiento

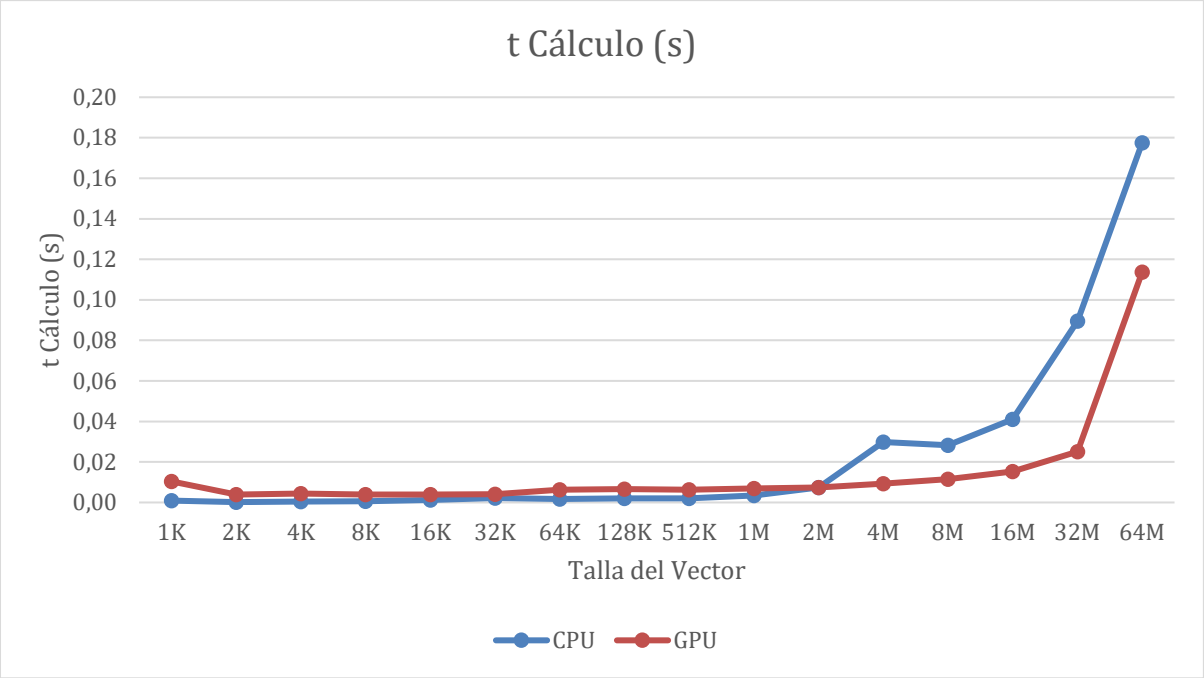
Y, la gráfica con los tiempos de comunicación Cliente – Servidor:



Gráfica 3.1.2: Tiempos de Comunicación

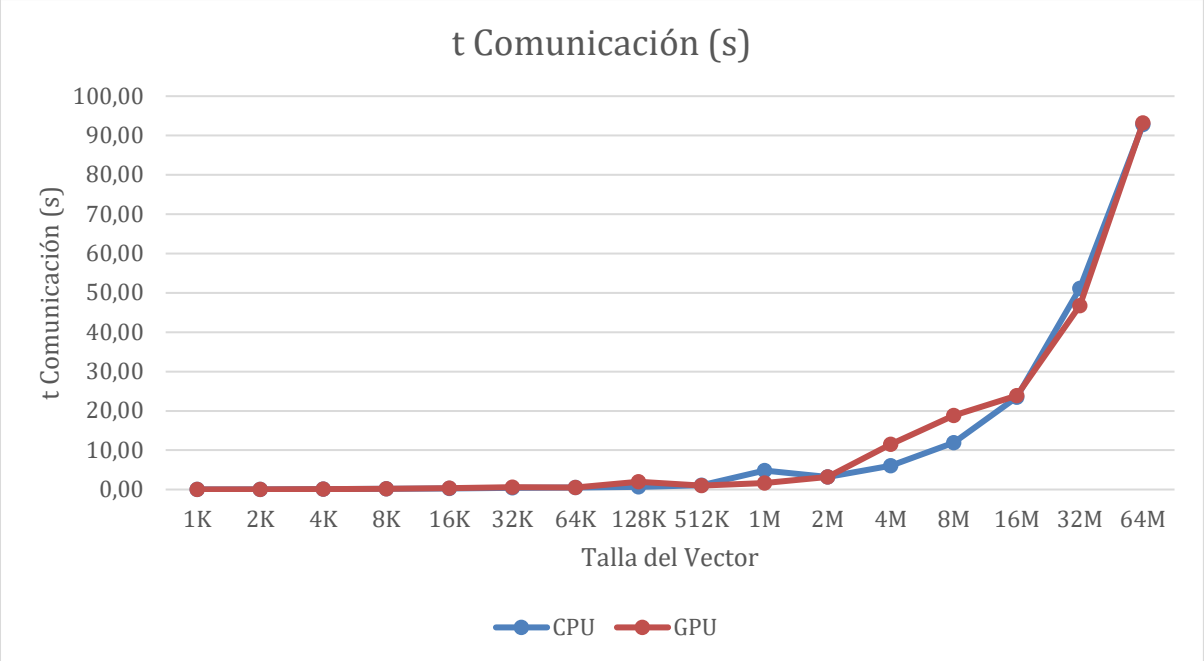
4.1.3.2. Experimento 3.2: Comparación temporal CPU-GPU de forma remota al servidor Cloud utilizando conexión LAN.

Se ha obtenido una tabla de resultados de tiempo de cálculo:



Gráfica 3.2.1: Tiempos de Procesamiento

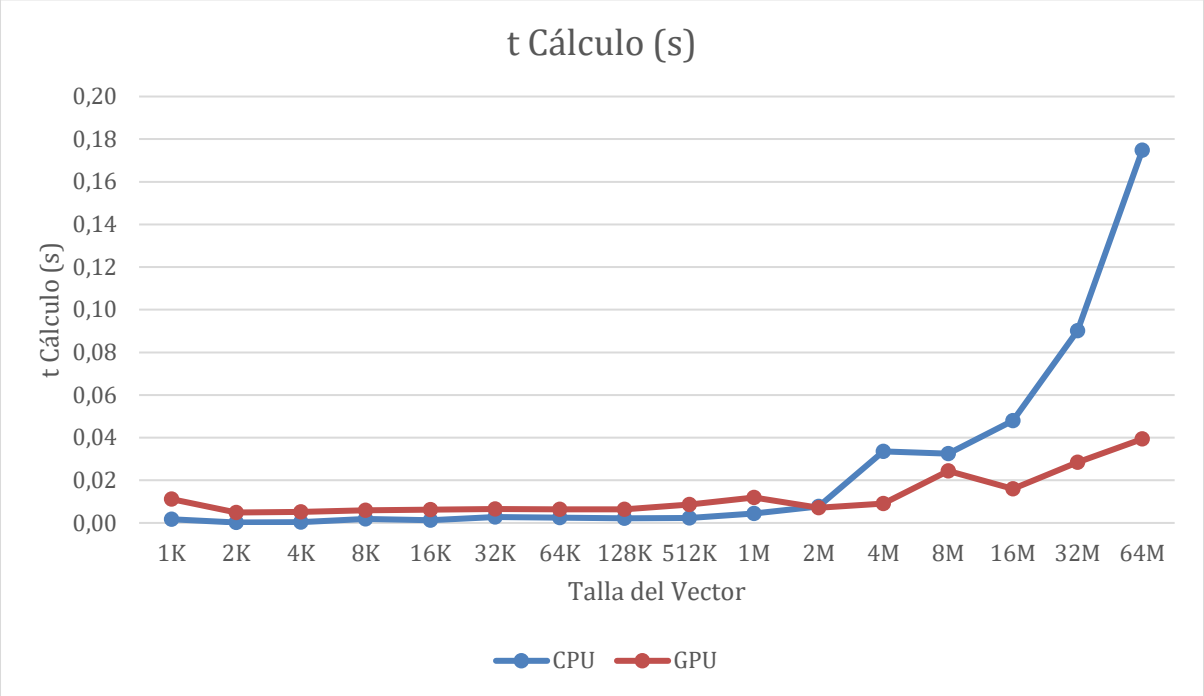
Y, la gráfica con los tiempos de comunicación Cliente – Servidor:



Gráfica 3.2.2: Tiempos de Comunicación.

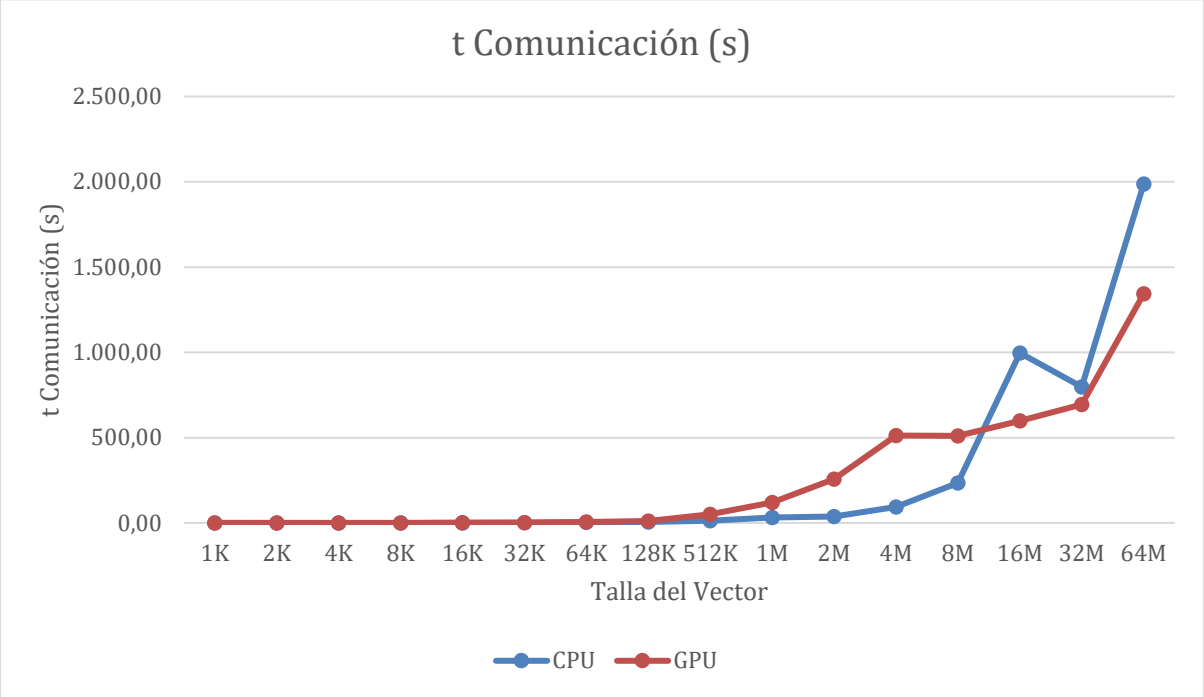
4.1.3.3. Experimento 3.3: Comparación temporal CPU-GPU de forma remota al servidor Cloud utilizando conexión 3G.

Este experimento se ha realizado 3 veces en 3 días diferentes en franjas de Mañana de 10:00 h a 12:00h, Tarde de 16:00 h – 18:00 h y Noche de 22:00 h – 00:00 h, con la finalidad de eliminar el factor carga de la red inalámbrica encontrado en algunas horas del día. Se ha llevado a cabo con unos porcentajes de cobertura del 60% al 65% por lo que la reproducción de este experimento en localizaciones distintas, así como con porcentajes de cobertura diferentes podría dar lugar a otros resultados difiriendo de los expuestos en este trabajo. Además, las gráficas mostradas corresponden con los datos obtenidos de la realización del promedio de los 3 resultados obtenidos (Mañana, Tarde y Noche) para el procesamiento en la CPU del Servidor y de los 3 resultados obtenidos (Mañana, Tarde y Noche) para el procesamiento en la GPU del Servidor.
Se ha obtenido una tabla de resultados de tiempo de cálculo:



Gráfica 3.3.1: Tiempos de Procesamiento

Y, la gráfica con los tiempos de comunicación Cliente – Servidor:



Gráfica 3.3.2: Tiempos de Comunicación.

4.1.4. Experimento 4: Comparación temporal CPU-GPU en una arquitectura Cloud real (Azure).

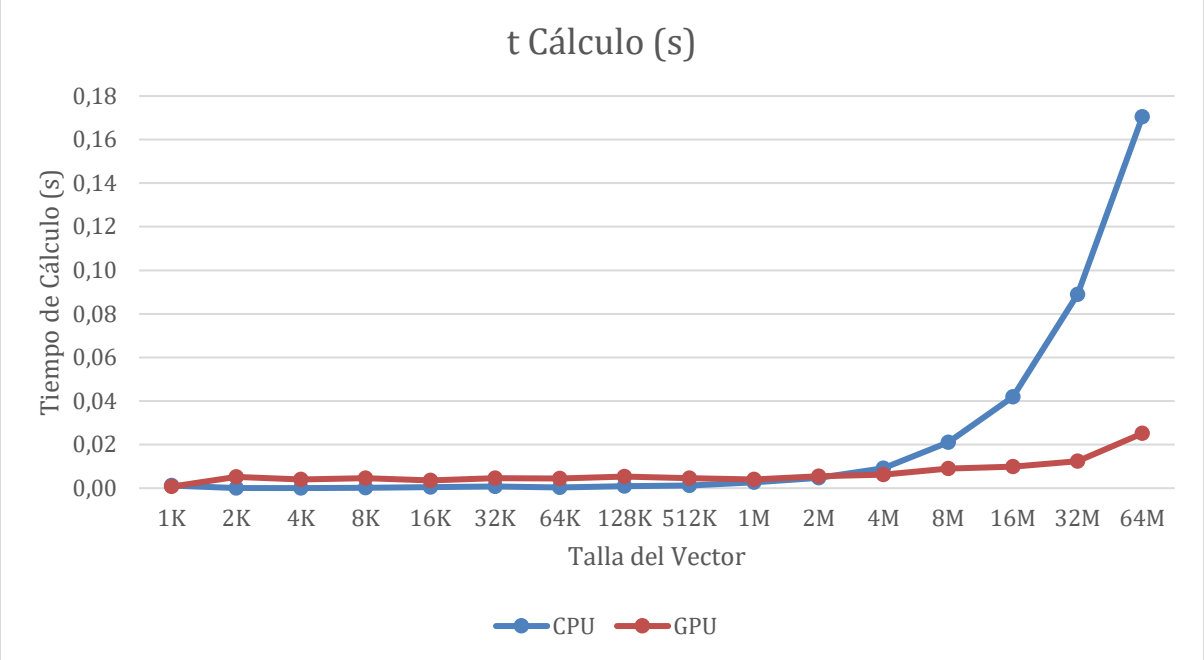
En este experimento se ha utilizado un ordenador como cliente y un servidor en el Cloud real como proporciona Azure. Para ello se ha creado una cuenta gratuita y se ha configurado una máquina virtual utilizando hardware adecuado que en este caso se engloba en la categoría de: *NC6 estándar (6 cpu, 56 GB de memoria)* detallado en el Anexo correspondiente.

Para esto se ha necesitado únicamente una IP pública para la conexión del socket cliente al servidor Cloud y proporcionada por Azure de Microsoft.

Finalmente, solamente detallar que los experimentos que se exponen a continuación han sido realizados mediante las mismas técnicas y finalidades de los experimentos del "*apartado 3.*" de este mismo trabajo únicamente con las conexiones del cliente mediante Wireless y mediante LAN y cuya principal finalidad es probar la idoneidad y las mejoras de utilizar una tarjeta gráfica de las últimas creadas por NVIDIA respecto de las utilizadas en otros experimentos expuestos en apartados anteriores.

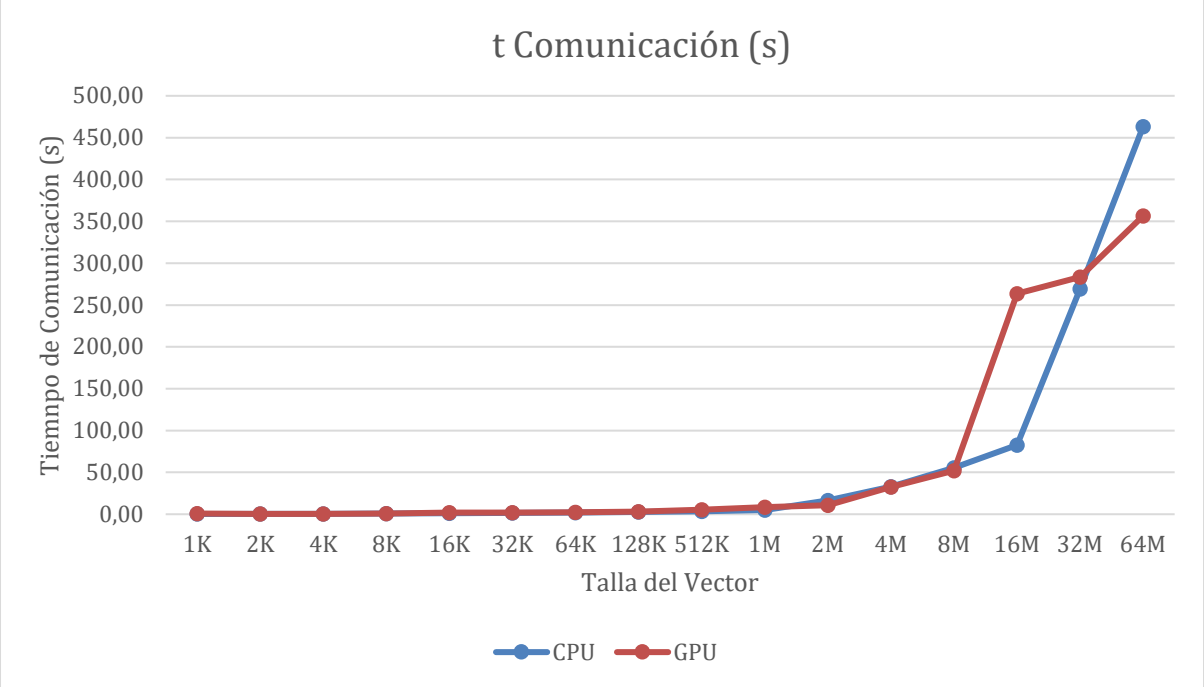
4.1.4.1. Experimento 4.1: Comparación temporal CPU-GPU de forma remota al servidor Cloud Azure utilizando conexión Wifi.

Se ha obtenido una tabla de resultados de tiempo de cálculo:



Gráfica 4.1.1: Tiempos de Procesamiento

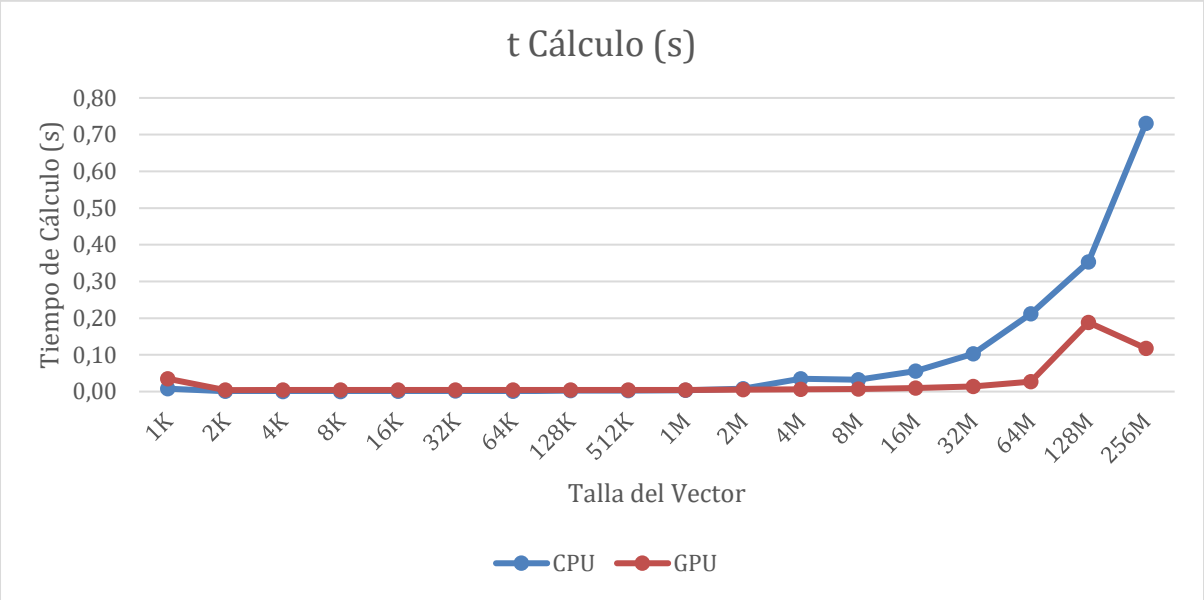
Y, la gráfica con los tiempos de comunicación Cliente – Servidor:



Gráfica 4.1.2: Tiempos de Comunicación

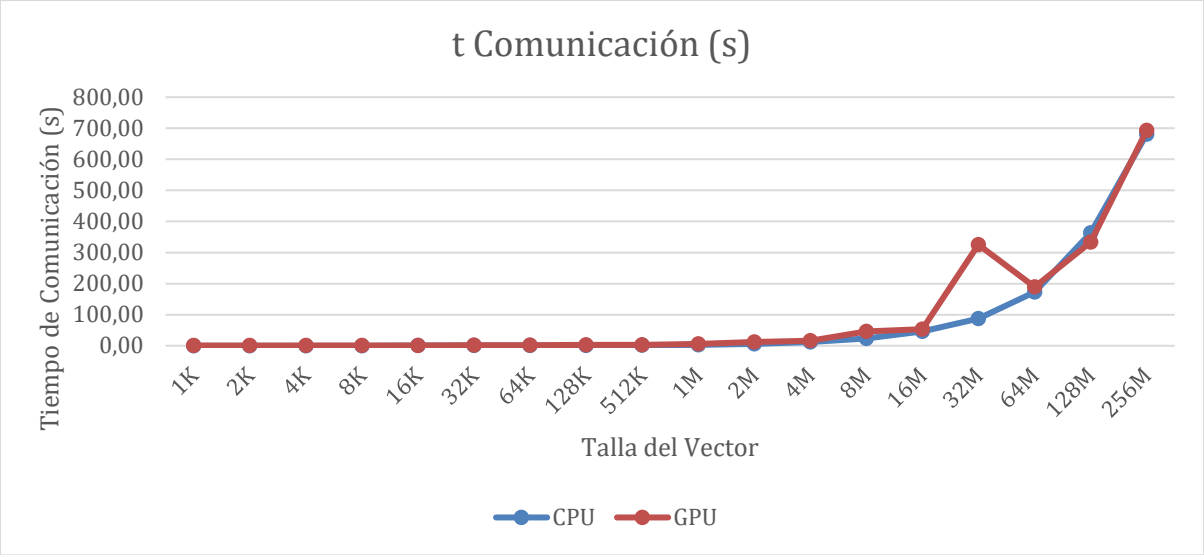
4.1.4.2. Experimento 4.2: Comparación temporal CPU-GPU de forma remota al servidor Cloud Azure utilizando conexión LAN.

Se ha obtenido una tabla de resultados de tiempo de cálculo:



Gráfica 4.2.1: Tiempos de Procesamiento

Y, la gráfica con los tiempos de comunicación Cliente – Servidor:



Gráfica 4.2.2: Tiempos de Comunicación

4.2. Pruebas de rendimiento

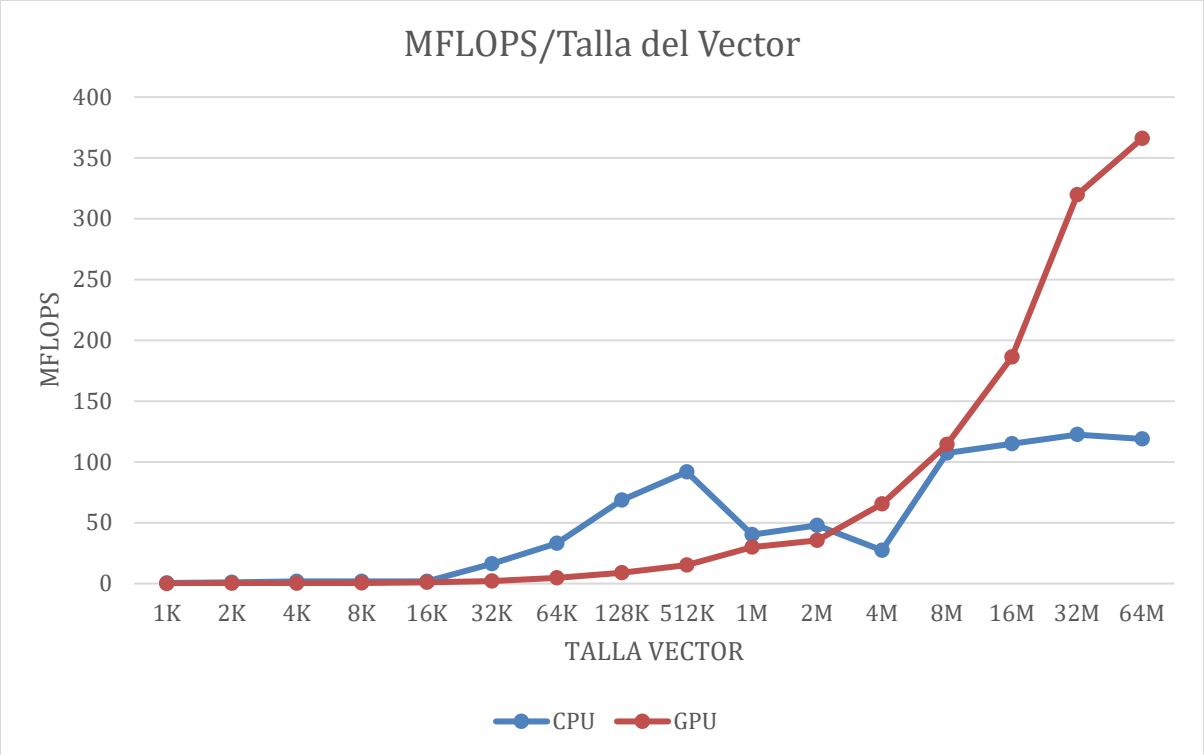
Seguidamente a los experimentos mostrados en este documento, se ha buscado la independencia de los resultados tratando de exportarlos a unas unidades que no dependan de variables externas como: procesos secundarios de los sistemas, temperatura, humedad, entre otras, con la que se han realizado los experimentos y con la que estos resultados sean útiles para comparar arquitecturas, tecnologías, etc.

Para esto, se ha elegido la unidad MFLOPS que se encarga de la medición del número de millones de operaciones de punto flotante por segundo, teniendo en cuenta el tiempo de ejecución transcurrido en realizar una operación en base a un algoritmo [15].
Se ha seguido una metodología basada en la ejecución del algoritmo propuesta para la suma de vectores, cuyo número de operaciones en coma flotante se corresponde con la talla del vector y posteriormente se ha realizado una división sencilla con el tiempo de ejecución elevado a 10^6 siguiendo la fórmula:

$$\text{MFLOPS} = \frac{\text{numero}_{\text{Operaciones}_{coma\,flotante}}}{\text{t}_{\text{cálculo}} * 10^6} = \frac{talla\ vector}{\text{t}_{\text{cálculo}} * 10^6}$$

4.2.1. MFLOPS Experimento 1:

Tras calcular os MFLOPS con la fórmula anterior, se ha obtenido un gráfico con los MFLOPS obtenidos para la visualización de los datos de una forma fácil y visual.:



Gráfica 5.1: MFLOPS/Talla del Vector. Experimento 1.

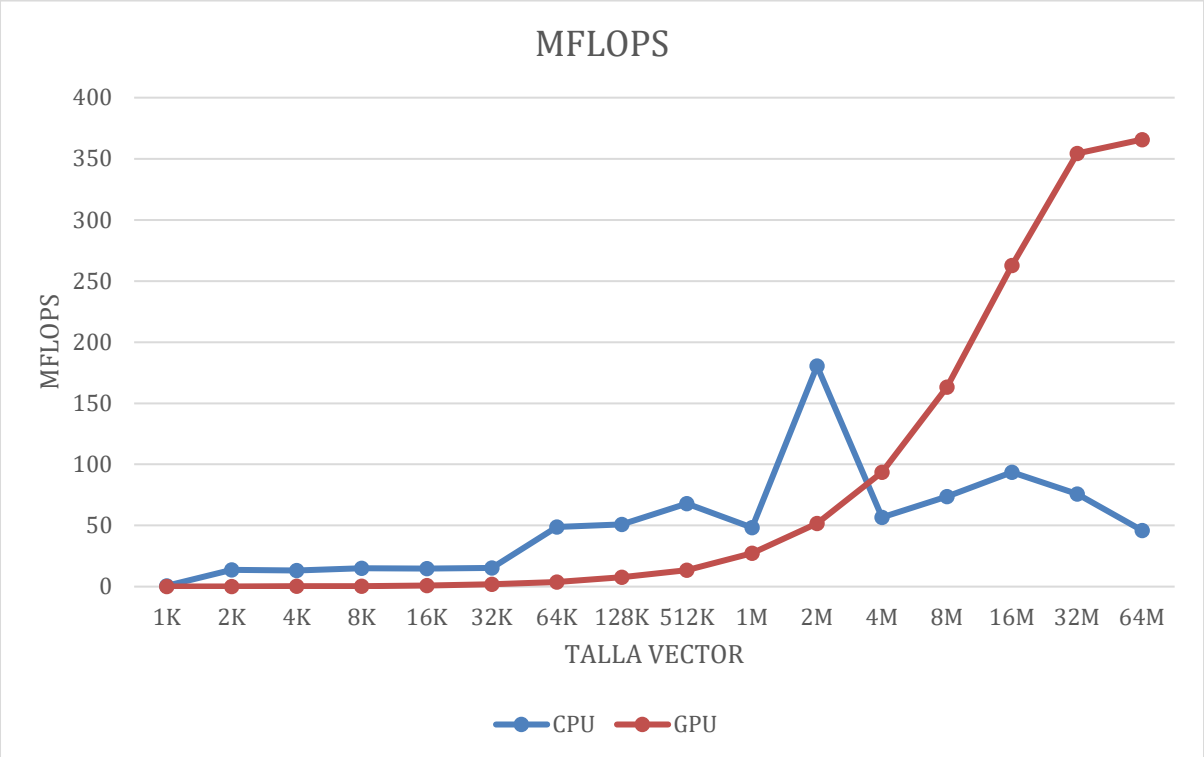
4.2.1.1. Conclusiones:

Como se puede ver en el gráfico, la creación de una arquitectura para la optimización de las operaciones mediante lenguajes paralelizables y con tarjetas de cálculo específico como las GPU, en nuestro caso con lenguaje CUDA, es muy recomendable debido al ínfimo tiempo de ejecución utilizado, como es posible ver debido a la diferencia de las dos líneas de tendencia comparando que el tiempo de ejecución en la CPU es mayor que el tiempo de ejecución en la GPU aprovechando la potencia de cálculo de la GPU para todos los cálculos y más notablemente en aquellos cálculos de más de 1.000.000 de componentes por vector.

4.2.2. MFLOPS Experimento 2:

4.2.2.1. MFLOPS Experimento 2.1:

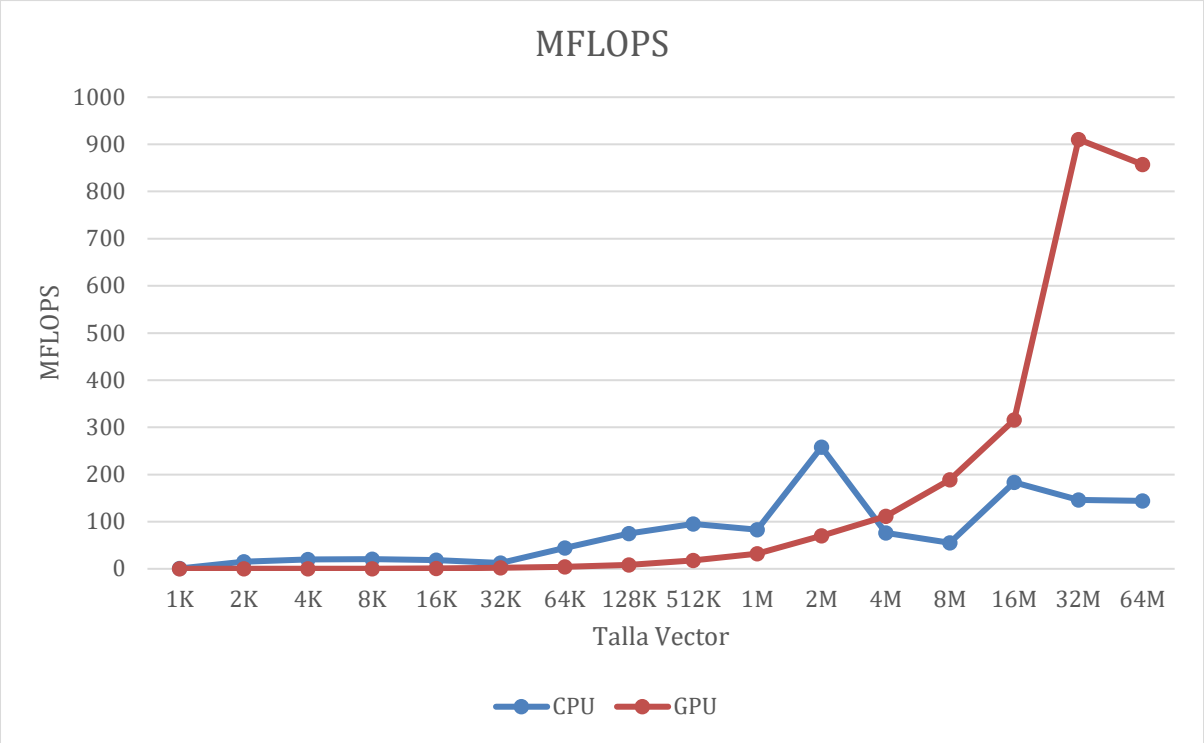
Tras calcular os MFLOPS con la fórmula anterior, se ha obtenido un gráfico con los MFLOPS obtenidos del experimento llevado a cabo mediante la conexión Wifi del cliente y el servidor para la visualización de los datos de una forma fácil y visual:



Gráfica 5.2: MFLOPS/Talla del Vector Experimento 2.1

4.2.2.2. MFLOPS Experimento 2.2:

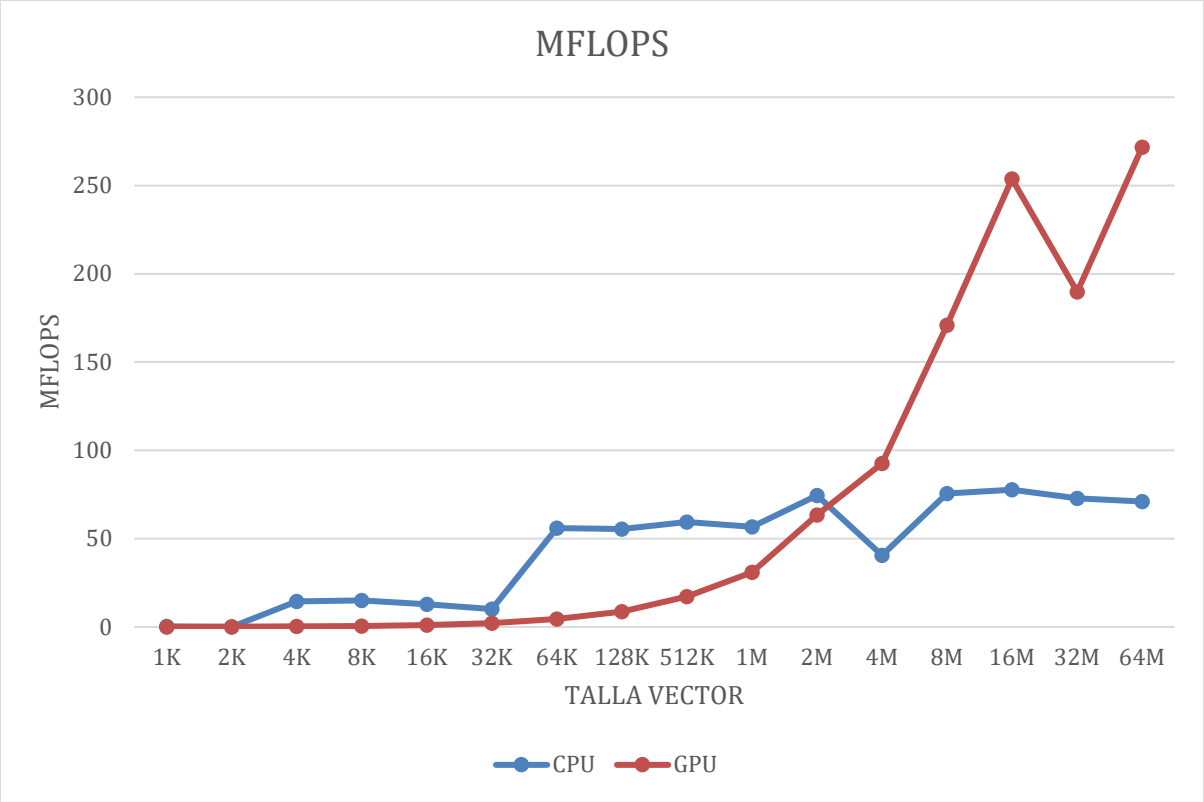
Tras calcular os MFLOPS con la fórmula anterior, se ha obtenido un gráfico con los MFLOPS obtenidos del experimento llevado a cabo mediante la conexión LAN del cliente y el servidor para la visualización de los datos de una forma fácil y visual:



Gráfica 5.3: MFLOPS/Talla del Vector Experimento 2.2

4.2.2.3. MFLOPS Experimento 2.3:

Tras calcular os MFLOPS con la fórmula anterior, se ha obtenido un gráfico con los MFLOPS obtenidos del experimento llevado a cabo mediante la conexión 3G del cliente y el servidor para la visualización de los datos de una forma fácil y visual:



Gráfica 5.4: MFLOPS/Talla del Vector Experimento 2.3

4.2.2.4. Conclusiones

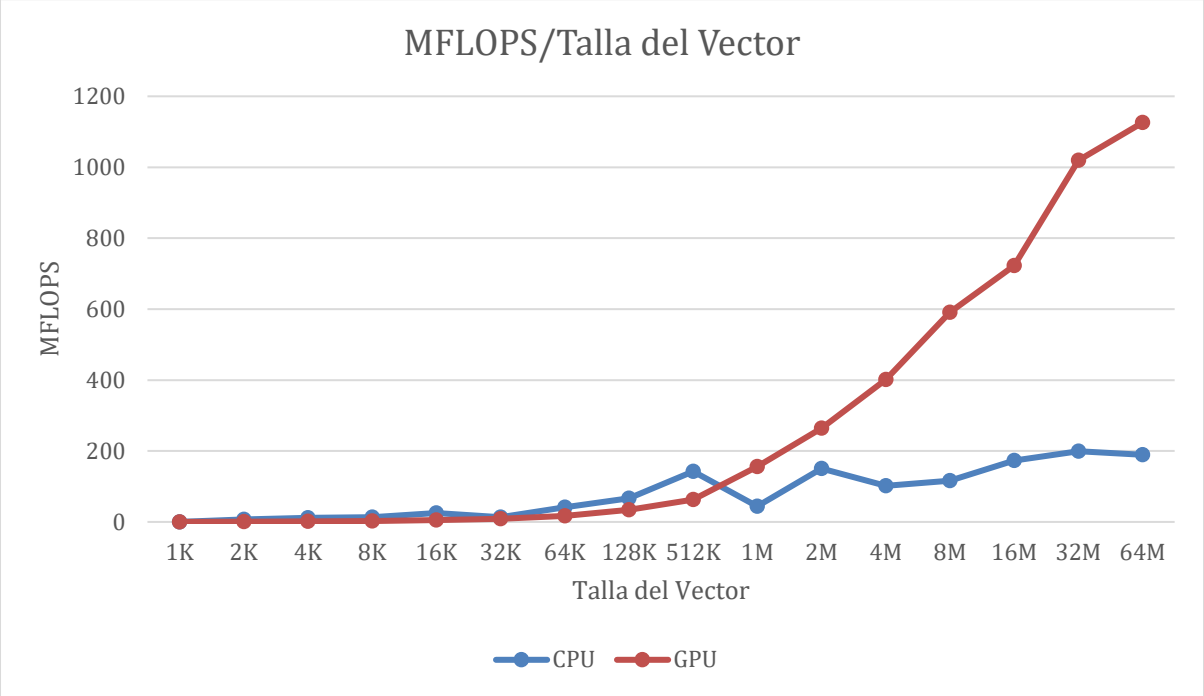
Tras observar las distintas gráficas obtenidas con los cálculos de los MFLOPS, se puede deducir que en los 3 experimentos arriba detallados, que, el cálculo de fórmulas matemáticas con un lenguaje capaz de soportar la paralelización así como la utilización y aprovechamiento de los diferentes núcleos encontrados en las tarjetas gráficas es recomendable para los cálculos de tamaño elevado como se puede mostrar en estos gráficos cuyo valor es significativo a partir de vectores de tamaño 2^{20} ya que comparando MFLOPS para el procesamiento en CPU y GPU, existe una amplia diferencia entre el cálculo secuencial (CPU) y el cálculo en paralelo (GPU) obteniendo un mejor valor debido a la inferioridad notable respecto del cálculo en la CPU aprovechando en un, aproximadamente, 11% de media la potencia de cálculo de la GPU para todos los cálculos realizados de dimensiones reducidas. También se puede observar el potencial del cálculo con GPU en vectores de tamaño considerable donde la CPU no puede llegar a competir por la abrumadora ventaja de la GPU y sus técnicas de paralelización masiva.

Para continuar, solamente será necesario tener en cuenta si es rentable la externalización a la nube puesto que el coste de comunicación, que en este caso dependerá del peso de los objetos a enviar y recibir del servidor, puede llegar a ser significativo tanto por el peso comentado anteriormente como factores ambientales si la conexión es inalámbrica, la tecnología de comunicación, además de otros factores como la carga existente en un dispositivo en el momento de envío de los elementos para su utilización en el servidor.

4.2.3. MFLOPS Experimento 3:

4.2.3.1. MFLOPS Experimento 3.1:

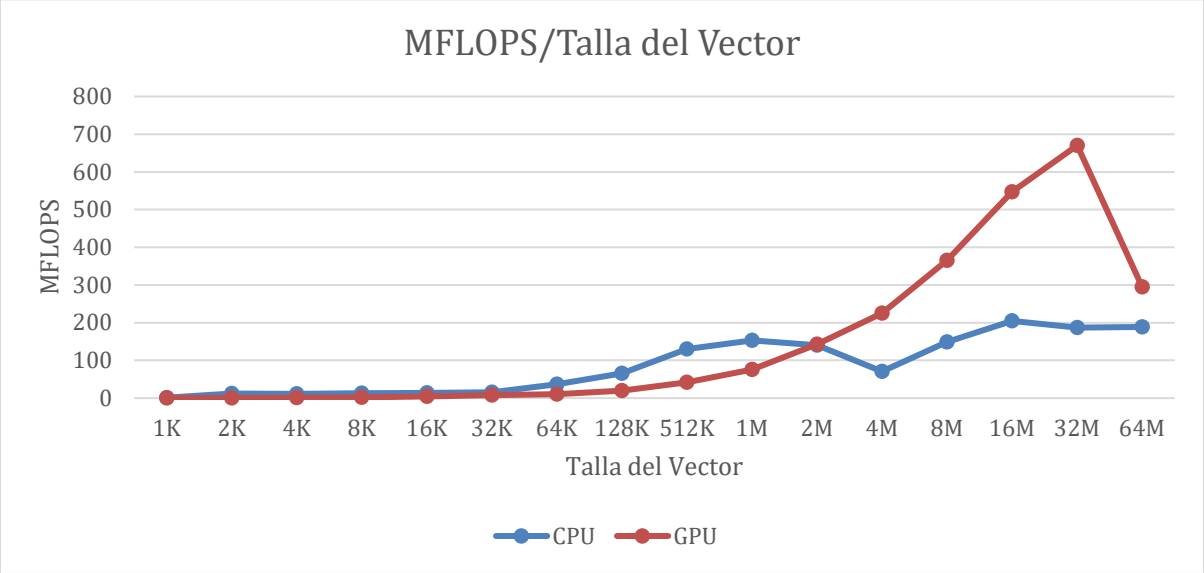
Tras calcular os MFLOPS con la fórmula anterior, se ha obtenido un gráfico con los MFLOPS obtenidos del experimento llevado a cabo mediante la conexión Wifi del cliente y el servidor mediante la conexión LAN para la visualización de los datos de una forma fácil y visual:



Gráfica 6.1: MFLOPS/Talla del Vector. Experimento 3.1.

4.2.3.2. MFLOPS Experimento 3.2:

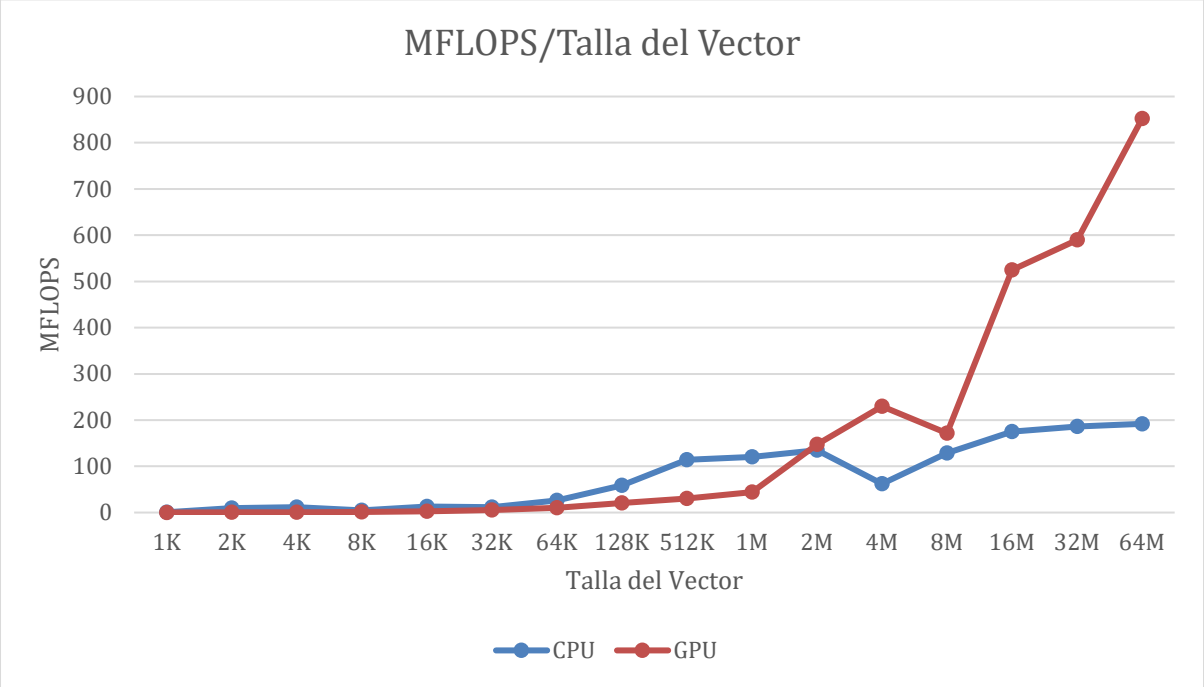
Tras calcular os MFLOPS con la fórmula anterior, se ha obtenido un gráfico con los MFLOPS obtenidos del experimento llevado a cabo mediante la conexión LAN del cliente y el servidor mediante la conexión LAN para la visualización de los datos de una forma fácil y visual:



Gráfica 6.2: MFLOPS/Talla del Vector. Experimento 3.2.

4.2.3.3. MFLOPS Experimento 3.3:

Tras calcular os MFLOPS con la fórmula anterior, se ha obtenido un gráfico con los MFLOPS obtenidos del experimento llevado a cabo mediante la conexión 3G del cliente y el servidor mediante la conexión LAN para la visualización de los datos de una forma fácil y visual:



Gráfica 6.3: MFLOPS/Talla del Vector. Experimento 3.3.

4.2.3.4. Conclusiones:

Tras la realización del experimento 3, con los 3 sub-experimentos, y observando las distintas gráficas obtenidas con los cálculos de los MFLOPS, se puede deducir que en los 3 experimentos arriba detallados, que, el cálculo de fórmulas matemáticas con un lenguaje capaz de soportar la paralelización así como la utilización y aprovechamiento de los diferentes núcleos encontrados en las tarjetas gráficas es recomendable para los cálculos de tamaño elevado como se puede mostrar en estos gráficos cuyo valor es significativo a partir de vectores de tamaño 2^{20} ya que comparando MFLOPS para el procesamiento en CPU y GPU, existe una amplia diferencia entre el cálculo secuencial (CPU) y el cálculo en paralelo (GPU) obteniendo un mejor valor debido a la inferioridad notable respecto del cálculo en la CPU aprovechando en un, aproximadamente, 10% de media la potencia de cálculo de la GPU para todos los cálculos realizados de dimensiones reducidas. También se puede observar el potencial del cálculo con GPU en vectores de tamaño considerable donde la CPU no puede llegar a competir por la abrumadora ventaja de la GPU y sus técnicas de paralelización masiva.

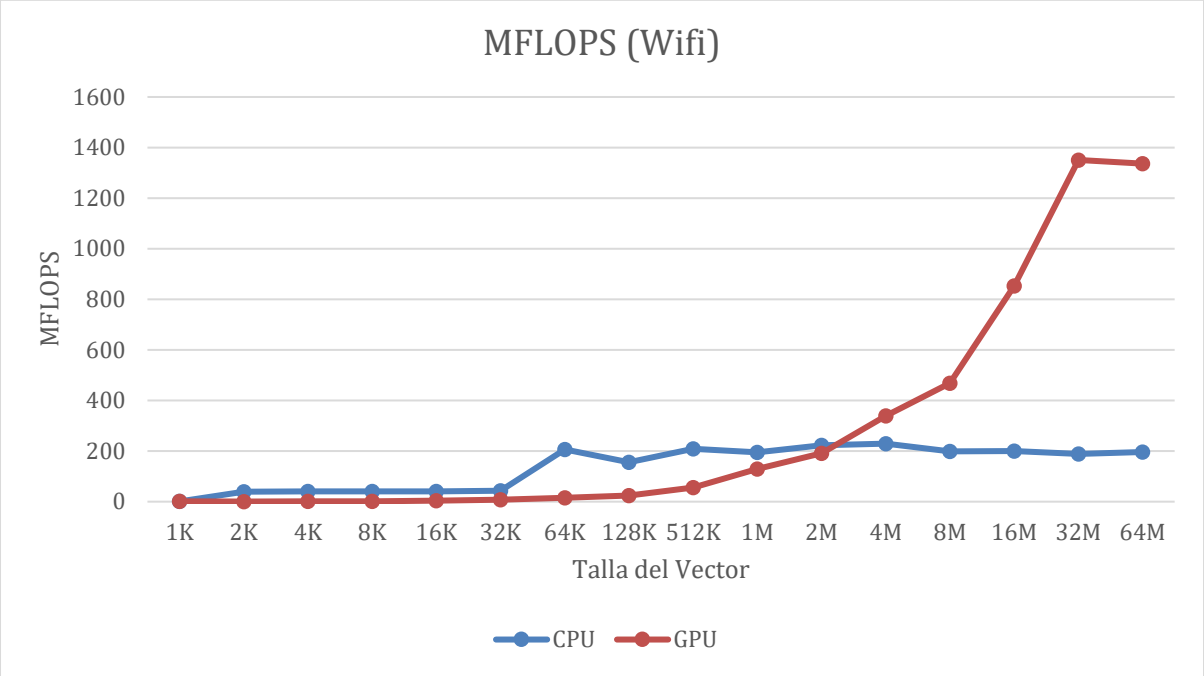
Y, solamente será necesario tener en cuenta si es rentable la externalización a la nube puesto que el coste de comunicación, que en este caso dependerá del peso de los objetos a enviar y recibir del servidor, puede llegar a ser significativo tanto por el peso comentado anteriormente como factores ambientales si la conexión es inalámbrica además de otros factores como la carga existente en un dispositivo en el momento de envío de los elementos para su utilización en el servidor.

Finalmente, será imprescindible un análisis de costes como mantenimiento, reparación entre otros puesto que la elección de un sistema como el propuesto en este experimento recaería totalmente en la empresa o institución que lo implante teniendo a cargo tanto el mantenimiento de este o futuros desarrollos que deban ser implantados para la puesta de marcha del sistema u optimizaciones del mismo.

4.2.4. MFLOPS Experimento 4:

4.2.4.1. MFLOPS Experimento 4.1:

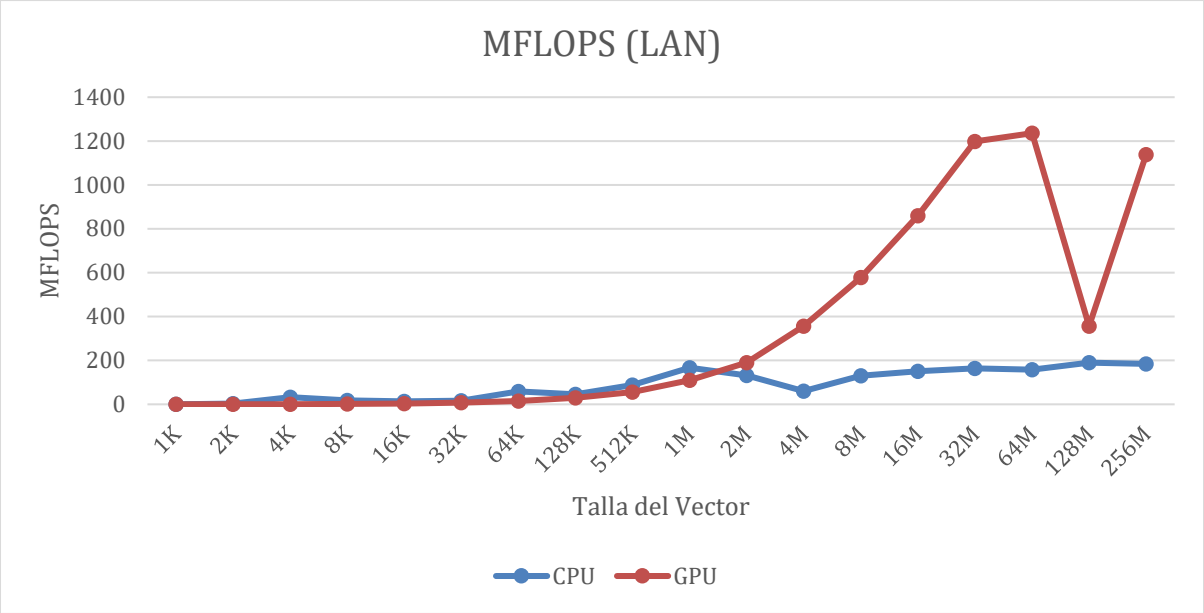
Tras calcular os MFLOPS con la fórmula anterior, se ha obtenido un gráfico con los MFLOPS obtenidos del experimento llevado a cabo mediante la conexión Wifi del cliente para la visualización de los datos de una forma fácil y visual:



Gráfica 7.1: MFLOPS/Talla del Vector. Experimento 4.1

4.2.4.2. MFLOPS Experimento 4.2:

Tras calcular os MFLOPS con la fórmula anterior, se ha obtenido un gráfico con los MFLOPS obtenidos del experimento llevado a cabo mediante la conexión LAN del cliente para la visualización de los datos de una forma fácil y visual:



Gráfica 7.2: MFLOPS/Talla del Vector. Experimento 4.2

4.2.4.3. Conclusiones:

Tras la realización del experimento 4, con los 2 sub-experimentos, y observando las distintas gráficas obtenidas con los cálculos de los MFLOPS, se puede deducir que en los 2 experimentos arriba detallados, que, el cálculo de fórmulas matemáticas con un lenguaje capaz de soportar la paralelización así como la utilización y aprovechamiento de los diferentes núcleos encontrados en las tarjetas gráficas es recomendable para los cálculos de tamaño elevado como se puede mostrar en estos gráficos cuyo valor es significativo a partir de vectores de tamaño 2^{18} ya que comparando MFLOPS para el procesamiento en CPU y GPU, existe una amplia diferencia entre el cálculo secuencial (CPU) y el cálculo en paralelo (GPU) obteniendo un mejor valor debido a la inferioridad notable respecto del cálculo en la CPU aprovechando en un, aproximadamente, 18% de media la potencia de cálculo de la GPU para todos los cálculos realizados de dimensiones reducidas. También se puede observar el potencial del cálculo con GPU en vectores de tamaño considerable donde la CPU no puede llegar a competir por la abrumadora ventaja de la GPU y sus técnicas de paralelización masiva.

También será necesario tener en cuenta si es rentable la externalización a la nube puesto que el coste de comunicación, que en este caso dependerá del peso de los objetos a enviar y recibir del servidor, puede llegar a ser significativo tanto por el peso comentado anteriormente como factores ambientales si la conexión es inalámbrica, la tecnología de comunicación, además de otros factores como la carga existente en un dispositivo en el momento de envío de los elementos para su utilización en el servidor.

Finalmente, será imprescindible un análisis de costes de la implantación de estos servicios en la nube; y tras utilizar en este experimento la nube comercial Azure por ser una empresa con una de las ofertas de procesamiento en la nube más completas y de menor precio para la realización de estas pruebas; para la elección correcta y completa para optimizar aún en mayor medida los resultados obtenidos así como para eliminar el coste de mantenimiento entre otros diluyendo estos costes en la cuota mensual desligando de la empresa suministradora de estos servicios los costes derivados de mantener el sistema entre otros.

Para acabar señalar que, utilizando la potencia de los cálculos de la GPU, por sus técnicas de paralelización masiva, y la utilización de éstas en un servidor en la nube con un procesador específico para servidores y de mayor potencia de los que habitualmente se comercializan, se ha logrado pasar la barrera de la suma de vectores con 2^{25} que no era posible con los dispositivos utilizados para los experimentos anteriores por lo que la externalización de software a servidores en la nube específicos con *hardware* bajo demanda hace de este tipo de programación muy recomendable para las empresas que necesiten de altas prestaciones en sus equipos debido a su actividad empresarial.

4.3. Prueba de utilización del hardware.

4.3.1. Prueba de funcionamiento en un escenario real.

Para continuar con el trabajo y redondear los resultados y conclusiones expuestas anteriormente, se ha creado un experimento con la finalidad de mostrar los resultados obtenidos tras ejecutar el software creado con un ordenador como servidor, localizado en los laboratorios de la Universidad de Alicante, y un numero n de ordenadores de un aula para la docencia.

4.3.1.1. Modus operandi.

El experimento se ha realizado mediante la compilación de los archivos necesarios para la ejecución del software creado. Para esto, se ha fijado una cantidad de 2^{10} elementos para los vectores a sumar (A y B) y, por tanto, devuelve un vector C de la misma talla.
Una vez compilado, se ha copiado en los ordenadores del aula, en adelante clientes, y se han ejecutado todos consecutivamente puesto que únicamente se podían ejecutar de cliente en cliente. Por esto no se puede hablar de una prueba de carga en sí puesto que no han sido todas las llamadas simultaneas, sino que con un pequeño *delay* provocado por el desplazamiento por el aula para ejecutar los archivos.

4.3.1.2. Resultados.

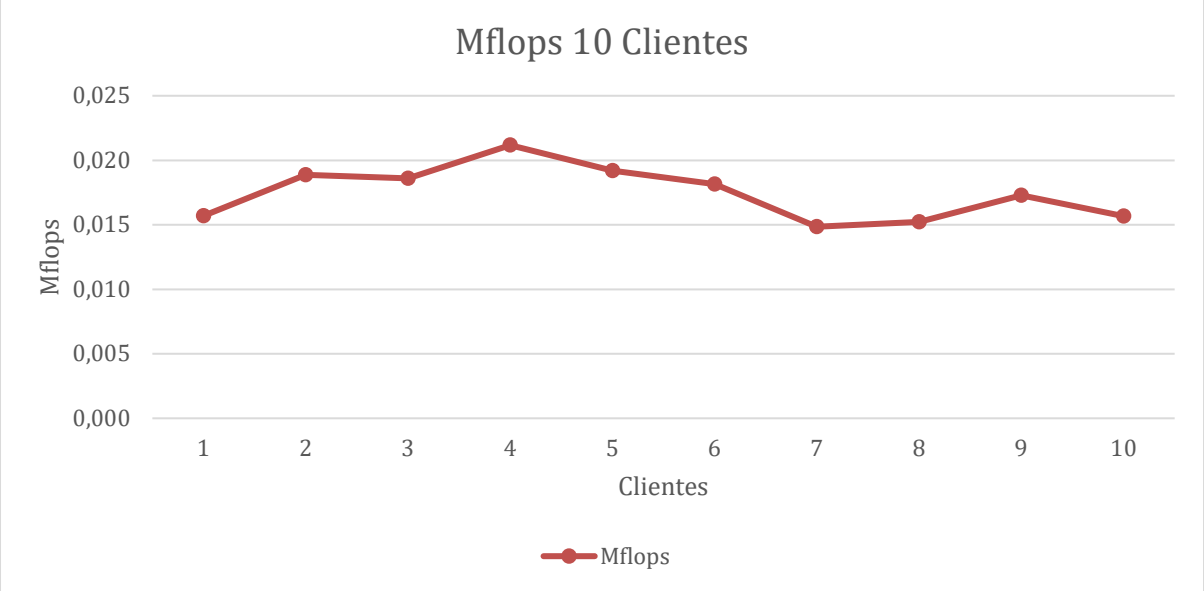
Tras la ejecución del experimento con 10 ordenadores como clientes y la ejecución del experimento con 16 ordenadores clientes realizando peticiones contra un servidor únicamente, cuyas especificaciones se pueden observar en el Anexo 4 de este documento, se han obtenido resultados tanto de tiempo de cálculo como tiempo utilizado en la comunicación de los clientes con el servidor.

4.3.1.2.1. Resultados Experimentación con 10 clientes.

Los resultados de comunicación se pueden ver en el Gráfico 8.1:

Gráfica 8.1: Tiempo de Comunicación para 10 clientes.

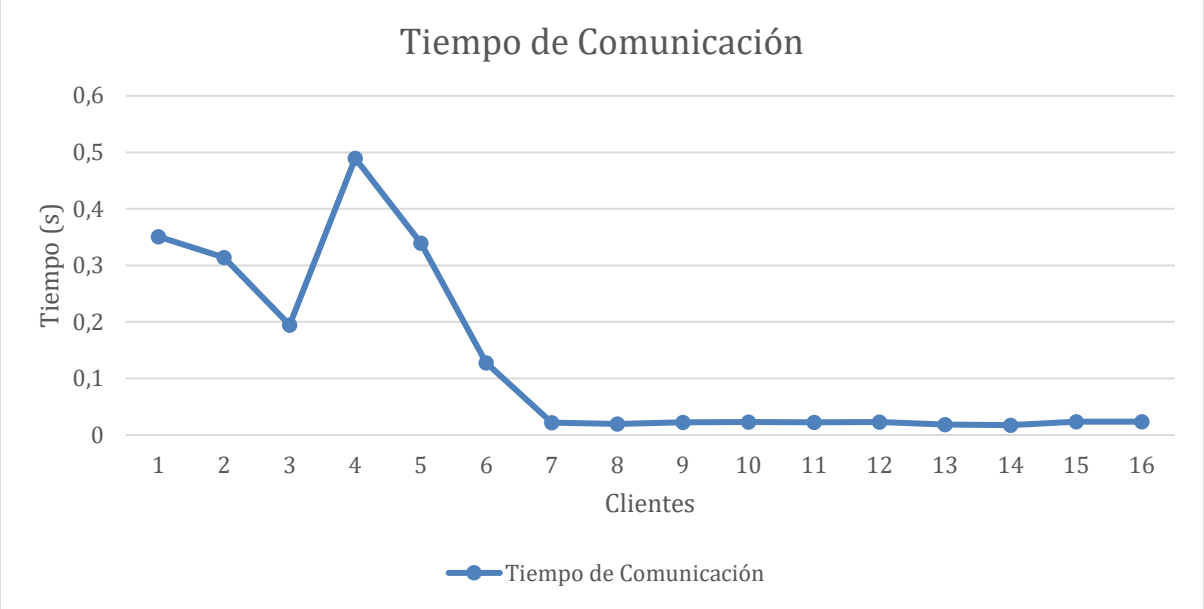
Los resultados del tiempo de cálculo se han convertido a MFLOPS con el objetivo de poder comparar con otras tarjetas gráficas si fuera necesario. Estos resultados se pueden ver en la Gráfica 8.2.



Gráfica 8.2: MFLOPS para 10 clientes.

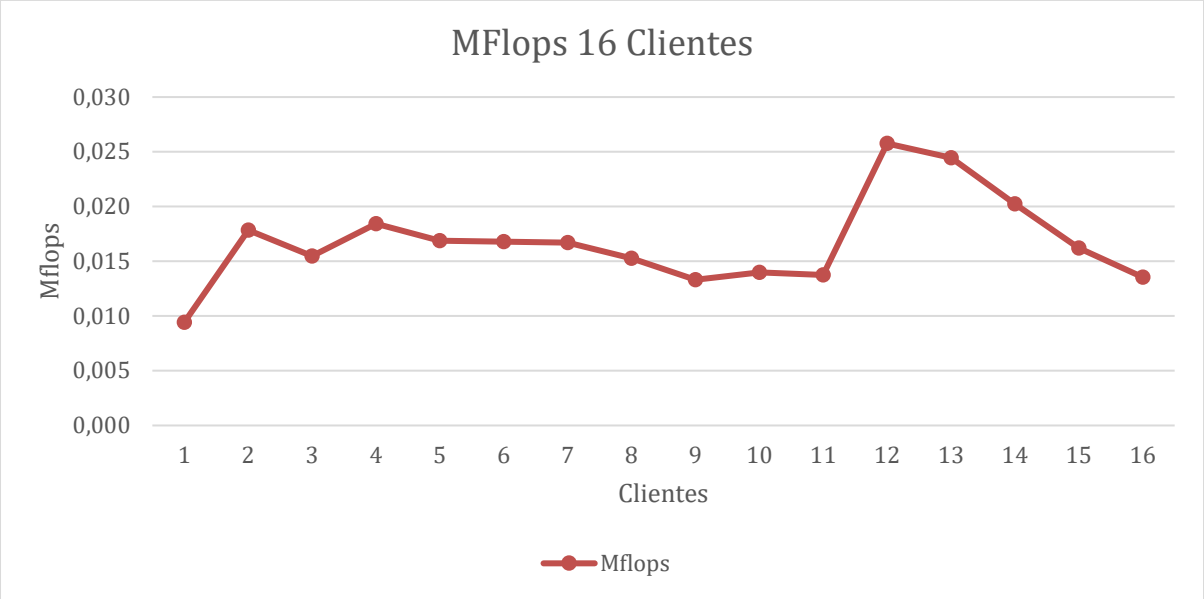
4.3.1.2.2. Resultados Experimentación con 16 clientes.

Los resultados de comunicación se pueden ver en el Gráfico 9.1:



Gráfica 9.1: Tiempo de Comunicación para 16 clientes.

Los resultados del tiempo de cálculo se han convertido a MFLOPS con el objetivo de poder comparar con otras tarjetas gráficas si fuera necesario. Estos resultados se pueden ver en la Gráfica 9.2.



Gráfica 9.2: MFLOPS para 16 clientes.

5. Conclusiones

Tras todo lo expuesto en esta memoria se puede afirmar que la elección de externalizar los cálculos gran complejidad proporciona ventajas como la reducción de costes debido a no necesitar hardware específico que puede llevar aparejados costes muy elevados y puede ser infrautilizada en determinadas situaciones y que no logre el rendimiento deseado.

Actualmente, con los desarrollos de 4G, 5G y fibra, los tiempos asociados debido al transporte de los Objetos a los servidores y clientes, serán mínimos y, posiblemente, despreciables según las velocidades de transmisión por la red.

La utilización de CUDA provee de técnicas de paralelización, ver Figura 6 donde ser observa la monitorización realizada para ver la utilización de la tarjeta gráfica representada por las rallas a la derecha de la imagen y que corresponden con los distintos clientes representados por la etiqueta "*Context X*" a la izquierda de la imagen en los distintos ejemplos mostrados anteriormente , y que junto a las técnicas de programación orientada a objetos y la creación de un servidor multihilo facilita gran cantidad de peticiones con las que dar respuesta a muchos clientes que demanden cálculos de gran complejidad en un tiempo muy reducido.

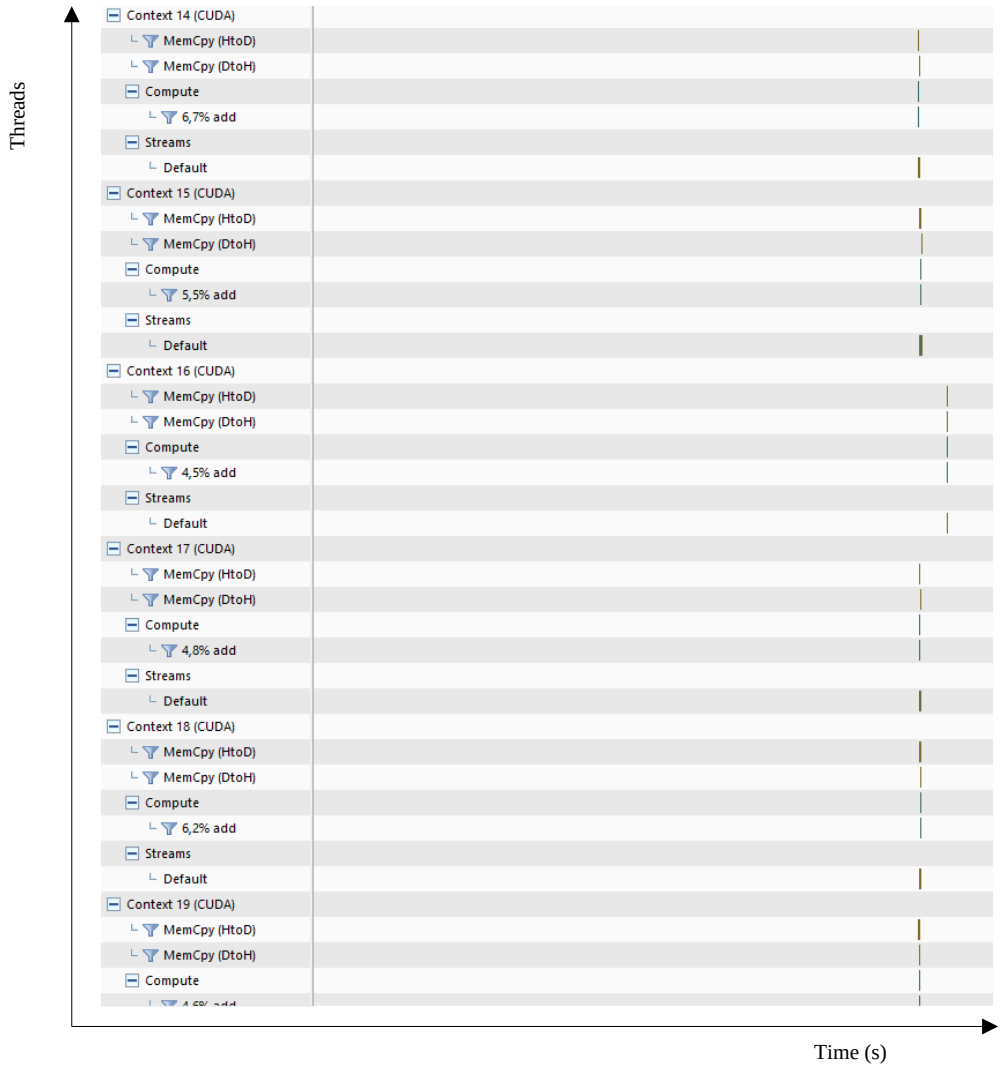


Fig. 6: Monitorización de un ejemplo de funcionamiento de la arquitectura propuesta.

Finalmente, se ha obtenido un conocimiento alrededor de esta forma de programar y se ha expuesto una serie de experimentos con los que se puede concluir la considerable mejora de rendimiento en los cálculos avanzados abriendo la puerta a invertir tiempo y dinero en el desarrollo de estas aplicaciones especializadas con las que una *GPU* cumpliría el cometido en un tiempo récord y con un rendimiento más elevado que si se utilizara la *CPU* del ordenador pudiendo ser insuficiente para los cálculos a realizar.

5.1. Futuros líneas de investigación y trabajo.

Tras la exposición del trabajo, y como la temática principal es innovadora se encuentran diversas líneas de investigación y trabajo como:

- Profundización en la optimización de las instrucciones CUDA y en las técnicas de paralelización de hilos mejorando el *kernel* de la tarjeta gráfica.
- Investigación y creación de nuevas formas de comunicación que disminuya el tiempo de espera minimizando así también la latencia en las tecnologías de la comunicación.
- Aplicación en distintas aplicaciones que necesiten calculo avanzado como la criptografía, simulación de fluidos, renderizado de imágenes, aplicaciones en inteligencia artificial, seguridad informática y ciberseguridad, entre otras aplicaciones.
- Investigación en ingeniería de *hardware* para obtener mejores dispositivos *GPU* con mayores prestaciones y de tamaño más reducido.
- Generalización de este tipo de programación puesto que la versión genérica actual de programación en tarjetas gráficas, *OpenGL*, no es tan preciso ni tan rápido como el proporcionado por CUDA.
- Creación de un sistema de licencias de nueva implantación, como las licencias por potencia de *GPU*, permitiendo a los usuarios pagar por lo que necesitan sin tener infrautilizado el software/hardware proporcionado.
- Ampliación de aplicaciones basadas en la nube de ventajas más que conocidas.

Anexos:

Anexo 1: Especificaciones ordenador cliente.

Hardware	Nombre
Procesador	AMD Ryzen R5 2400G 3.6GHZ
Memoria RAM	Kingston HyperX Fury Black DDR4 2400 PC4-19200 8GB CL15
SSD	Kingston A400 SSD 240GB
Tarjeta Gráfica dedicada	MSI Radeon RX 550 AERO ITX 4G OC GDDR5
Especificaciones de Conectividad	Realtek 8111H

Anexo 2: Especificaciones tarjeta gráfica ordenador servidor MSI.

Hardware	Nombre
Tarjeta gráfica:	NVIDIA GeForce GTX 1050 2GB GDDR5.
Arquitectura de GPU:	Pascal
Núcleos de CUDA:	640
Reloj del núcleo:	1354 MHz
Reloj del núcleo Acelerado:	1455 MHz
Velocidad de datos de la memoria:	7.01 Gbps
Interfaz de memoria:	128-bits
Ancho de banda de la memoria:	112.13 GB/s
Memoria total disponible para gráficos:	6102 MB
Memoria de vídeo dedicada:	2048 MB GDDR5
Memoria compartida del sistema:	4054 MB
Bus:	PCI Express x16 Gen3

Anexo 3: Especificaciones ordenador servidor MSI.

Hardware	Nombre
Procesador	Kabylake i7-7700HQ+HM175 (3.8GHz, 6MB)
Memoria RAM	8GB DDR4-2400 SODIMM
SSD	Sí. Conexión M.2 SSD Combo (NVMe PCIe Gen3 x4 / SATA)
Tarjeta Gráfica dedicada	NVIDIA GeForce GTX 1050, 2GB GDDR5 *
Especificaciones de Conectividad	Gb LAN, 802.11 AC

(*) NVIDIA GeForce GTX 1050 especificada en el anexo correspondiente.

Anexo 4: Especificaciones servidor Cloud simulado en el laboratorio de la Universidad de Alicante.

Hardware	Nombre
Procesador	Intel Pentium G850 2.90 GHz
Memoria RAM	8,00 GB
SSD	No. 500 GB HDD
Tarjeta Gráfica dedicada	NVIDIA GeForce GTX 480, 1.5GB GDDR5 *
Especificaciones de Conectividad	Gb LAN, 802.11 AC

(*) NVIDIA GeForce GTX 480 especificada en el anexo correspondiente.

Anexo 5: Especificaciones tarjeta gráfica servidor Cloud simulado en el laboratorio de la Universidad de Alicante.

Hardware	Nombre
Tarjeta gráfica:	NVIDIA GeForce GTX 480, 1.5GB GDDR5
Arquitectura de GPU:	Fermi
Núcleos de CUDA:	480
Reloj del núcleo:	1401 MHz
Reloj del núcleo Acelerado:	1848 MHz
Interfaz de memoria:	384 bits
Ancho de banda de la memoria:	177.4 GB/s
Memoria total disponible para gráficos:	1536 MB GDDR5
Bus:	PCIe 2.0 x16

Anexo 6: Especificaciones de la máquina virtual en Azure.

Categoría contratada: *NC6 estándar (6 cpu, 56 GB de memoria)*

Hardware	Nombre
Procesador	(6 cores) Intel E5-2690v3
Memoria RAM	56 GB
SSD	380 GB SSD
Tarjeta Gráfica dedicada	Sí, NVIDIA Tesla K80 *
Especificaciones de Conectividad	Gb LAN, 802.11 AC

(*) NVIDIA Tesla K80 especificada en el anexo correspondiente.

Anexo 7: Especificaciones tarjeta gráfica máquina virtual Azure.

Categoría contratada: *NC6 estándar (6 CPU, 56 GB de memoria)*

Hardware	Nombre
Tarjeta gráfica:	NVIDIA Tesla K80
Arquitectura de GPU:	Kepler
Núcleos de CUDA:	4.992 (2x2496)
Reloj del núcleo:	5 GHz
Ancho de banda de la memoria:	768 bit (2x384-bit)
Memoria total disponible para gráficos:	24GB GDDR5 (2x12GB)

Referencias.

1. Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications, 1(1):7-18, May 2010.

2. Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. ACM SIGCOMM Computer Communication Review, 38(4):63, oct 2008.

3. Jerry Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. Commun. ACM, 51(1):107113, January 2008.

4. Google Inc. Google app engine. https://console.cloud.google.com. Último acceso: 18/11/2018.

5. Amazon Inc. Tipos de cloud computing. https://aws.amazon.com/es/types-of-cloud-computing/. Último acceso: 18/11/2018.

6. Kenji E. Kushida, Jonathan Murray, and John Zysman. Diffusing the cloud: Cloud computing and implications for public policy. Journal of Industry, Competition and Trade, 11(3):209237, Sep 2011. Último acceso: 24/11/2018.

7. Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research. Último acceso: 27/11/2018.

8. Tanenbaum, A.: Sistemas Distribuidos. Último acceso: 02/12/2018.

9. Quang Hieu Vu, Mihai Lupu, Beng Chin Ooi. "Peer-to-Peer Computing. Principles and Applications". Springer. Último Acceso: 03/12/2018.

10. M. S. Gordon, D. K. Hong, P. M. Chen, J. Flinn, S. Mahlke, and Z. M. Mao, "Accelerating mobile applications through flip-flop replication," in Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services, ser. MobiSys '15. New York, NY, USA: ACM, 2015, pp. 137–150.

11. Zhou, Bowen. "Code Offloading and Resource Management Algorithms for Heterogeneous Mobile Clouds". Último acceso: 02/01/2019.

12. NVIDIA: Cuda parallel computing platform. http://developer.nvidia.com/category/zone/cuda-zone. Último acceso: 15/01/2018.

13. NVIDIA: Ptx: Parallel thread execution isa. http://developer.nvidia.com/cuda/nvidia-gpu-computing-documentation. Último acceso: 15/01/2018.

14. Lee, C., Ro, W.W. & Gaudiot, JL. Int J Parallel Prog (2014) 42: 384. https://doi.org/10.1007/s10766-013-0252-y. Último acceso: 15/01/2018.

15. C. Martínez, E. Castro. "Proyecto Intermedio Herramientas Computacionales: Medición de MFLOPS en problemas Matriciales". Octubre 2017. Último acceso: 25/02/2019.

16. H Mora, J Peral, A Ferrández, D Gil, J Szymanski, Distributed Architectures for Intensive Urban Computing: A Case Study on Smart Lighting for Sustainable Cities, IEEE Access 7, 58449-58465, 2019.

17. H Mora, FJ Mora Gimeno, MT Signes-Pont, B Volckaert, Multilayer Architecture Model for Mobile Cloud Computing Paradigm, Complexity, 2019.

18. H Mora, M Signes-Pont, D Gil, M Johnsson, Collaborative working architecture for IoT-based applications, Sensors 18 (6), 1676, 2018

19. H Mora, JF Colom, D Gil, A Jimeno-Morenilla, Distributed computational model for shared processing on Cyber-Physical System environments, Computer Communications 111, 68-83, 2017.

20. JF Colom, D Gil, H Mora, B Volckaert, AM Jimeno, Scheduling framework for distributed intrusion detection systems over heterogeneous network architectures, Journal of Network and Computer Applications 108, 76-86, 2018