UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS ESPECÍFICAS DE
TELECOMUNICACIÓN

ESTUDIO DEL TOOLBOX "*PARALLEL COMPUTING*" DE MATLAB®. APLICACIÓN A MÉTODOS DE PROCESADO DE SEÑAL

Autor:
D. Pablo Marina Boillos
Tutores:
Dr. D. Carlos Gómez Peña

Dr. D. Jesús Poza Crespo
Valladolid, 11 de junio de 2018

TÍTULO:	**Estudio del Toolbox "*Parallel Computing*" de MATLAB®. Aplicación a Métodos de Procesado de Señal**
AUTOR:	**D. Pablo Marina Boillos**
TUTORES:	**Dr. D. Carlos Gómez Peña**
	Dr. D. Jesús Poza Crespo
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**Dr. Dña. María García Gadañón**
SECRETARIO:	**Dr. D. Carlos Gómez Peña**
VOCAL:	**Dr. D. Jesús Poza Crespo**
SUPLENTE 1:	**Dr. D. Roberto Hornero Sánchez**
SUPLENTE 2:	**Dr. D. Miguel López-Coronado Sánchez-Fortún**

Donde hay voluntad, hay un camino.

RESUMEN

El procesado de señales biomédicas se ha convertido en los últimos años en una de las ramas más destacadas de la Ingeniería Biomédica, que ha permitido evolucionar desde una evaluación visual de las características de las señales biomédicas hasta el uso de sistemas de análisis y procesado avanzados en la actualidad.

Dicho avance ha propiciado que los requisitos computacionales sean cada vez más exigentes, desembocando en una mayor cantidad de datos generados para su posterior procesamiento y análisis. Este aumento de información ha causado que su tiempo de procesamiento aumente de forma considerable y que cada vez se requieran de sistemas más potentes.

La computación paralela permite solventar parcialmente este problema. Básicamente, consiste en dividir una tarea compleja en varias subtareas para que sean ejecutadas por lotes al mismo tiempo, repartiéndolas entre los procesadores disponibles. Con ello, no solo se intenta reducir el tiempo de realización de la tarea inicial, sino que también se intenta aportar escalabilidad al problema.

En este Trabajo Fin de Grado (TFG) se han aplicado técnicas de computación paralela a dos programas en MATLAB® desarrollados por el Grupo de Ingeniería Biomédica de la Universidad de Valladolid con una gran carga computacional, para intentar reducirla y obtener un coste de ejecución más aceptable.

Entre las distintas estrategias utilizadas, destacaron la vectorización y los bucles paralelos, que conseguían reducir hasta en un 65% la carga computacional. Ambas estrategias destacan por su facilidad de uso y su alta eficacia.

PALABRAS CLAVE

Computación paralela, tiempo de ejecución, *Parallel Computing Toolbox*, procesador, modelo de paralelización, carga computacional

ABSTRACT

Biomedical signal processing is an important field on Biomedical Engineering, which has evolved from a simple visual evaluation of the biomedical signals to advanced analysis and processing algorithms.

As the computational requirements are nowadays more demanding and the amount of data that has to be analyzed is huge, powerful systems are needed.

Parallel Computing enables us to solve partially this problem. A complex processing task is splitted into small chunks in order to be executed simultaneously, by sending them to different processors, diminishing the computational cost time and making the system scalable.

In this Degree Final Project, Parallel Computing techniques have been applied to two MATLAB® programs, developed by the Biomedical Engineering Group of the University of Valladolid, with a very high computational cost, with the aim of reducing the processing time.

Among the different strategies analyzed, vectorization and parallel for loops relieved computational charge, so these ones have been selected as the best strategies for our scripts, reducing computational charge by 65%.

KEYWORDS

Parallel Computing, execution time, Parallel Computing Toolbox, processor, Parallel Computing model, computational charge

AGRADECIMIENTOS

En primer lugar, mi más sincera gratitud a mis tutores Jesús Poza Crespo y Carlos Gómez Peña, por haberme orientado durante todo el camino en la realización de este Trabajo Fin de Grado, ya que sin su apoyo no podría haberlo llevado a cabo.

Mis agradecimientos también van para el Grupo de Ingeniería Biomédica de la Universidad de Valladolid, especialmente a Roberto, por haberme echado una mano en caso de necesidad, por sus consejos y por todos los buenos ratos que hemos compartido juntos.

Gracias a Chiara, Cristina, Tonia, Jorge, Víctor y Adrián, por estar ahí.

Gracias a mis padres, por su apoyo y ánimo, y a todos mis compañeros, por todos estos años de vivencias y buenos momentos.

Mil gracias.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. COMPUTACIÓN PARALELA

En los últimos años, el rápido desarrollo de las computadoras ha hecho que estas sean capaces de procesar cantidades de datos cada vez más ingentes. Sin embargo, el aumento del volumen de información a procesar hace que el retardo necesario también aumente, en algunos casos de forma exponencial, de tal forma que en determinadas situaciones ni siquiera se pueda llegar a completar la tarea en condiciones normales. La computación paralela estudia de qué formas se puede dividir un programa con grandes requerimientos de tiempo y memoria en un conjunto de tareas, de tal forma que estas sean ejecutadas por lotes al mismo tiempo entre varios procesadores, con el fin de reducir su carga computacional y, por tanto, reducir su tiempo de ejecución, así como aportar escalabilidad a nuestro sistema (Vajteršic et al., 2009). En este Trabajo de Fin de Grado (TFG) se estudiarán algunas arquitecturas de computación paralela, así como algunos métodos disponibles en MATLAB®, aplicándolos a varios casos reales.

1.2. OBJETIVOS DEL TFG

El objetivo principal de este Trabajo Fin de Grado consiste en paralelizar dos códigos de MATLAB® usados por el Grupo de Ingeniería Biomédica (GIB) de la Universidad de Valladolid (UVA) con el fin de reducir su carga computacional y, por ende, su tiempo de ejecución. Para ello, se establece una serie de objetivos secundarios:

- Identificación de varias estrategias de paralelización que sean posibles de aplicar con la herramienta MATLAB®, usando su *toolbox* de computación paralela (*Parallel Computing Toolbox*).
- Identificación de zonas en el código que produzcan una sobrecarga en el procesador y que, por tanto, sean susceptibles de ser paralelizadas.
- Implementación de las estrategias recogidas en nuestra lista de estrategias y determinación de la viabilidad de cada una.
- Comparativa de cada una de las estrategias anteriores y elección de aquellas más convenientes.

1.3. METODOLOGÍA

La metodología que se ha desarrollado para la consecución de los objetivos anteriormente mencionados es la siguiente:

- Fase de familiarización con los siguientes *toolboxes* de MATLAB®:
  - *Parallel Computing Toolbox*
  - *GPU Computing*
  - *Signal Proccesing Toolbox*
  - *Wavelet Toolbox*

1. INTRODUCCIÓN

- Fase de documentación:
  - Estudio de las propiedades de los electroencefalogramas (EEG) y magnetoencefalogramas (MEG).
  - Familiarización con los estudios de procesado de señal del GIB de la UVA.
  - Consulta de la documentación oficial de MATLAB® para la comprensión del funcionamiento de aquellas funciones necesarias.
- Fase de identificación y catalogación de las estrategias de paralelización aplicables a nuestro caso, en función de los recursos y funciones disponibles en MATLAB®.
- Fase de paralelización de los dos códigos de MATLAB® del GIB de la UVA:
  - Identificación de las partes y funciones en los programas del GIB de la UVA susceptibles de ser paralelizadas, en base a aquellas zonas con tiempo de ejecución elevado.
  - Implementación y prueba de las estrategias recogidas en el punto anterior y elección de aquella más adecuada en cada caso.
  - Batería de pruebas de verificación.
- Fase de recopilación y análisis de resultados
  - Comparativa de tiempos entre distintas versiones.
  - Discusión y análisis de resultados.
  - Conclusiones.

2. INTRODUCCIÓN AL PROCESADO DE SEÑALES BIOMÉDICAS

Una señal biomédica se define como la variación de la corriente eléctrica u otra magnitud generada en un sistema biológico y que se utiliza con fines diagnósticos. Para ello, deben de registrarse en algún tipo de soporte de tal forma que el parámetro a medir se recoja de la manera más precisa posible (Martín González, 2015; Poza, 2008). El procesado de señales biomédicas constituye una rama de la Ingeniería Biomédica que aplica los principios de ingeniería para, por una parte, entender, modificar o controlar sistemas biológicos y, por otra, diseñar y fabricar sistemas que monitoricen las funciones fisiológicas así como contribuir al diagnóstico y tratamiento de los pacientes (Bronzino, 2006).

El procesado de señales biomédicas se divide en tres etapas que, a su vez, se subdividen en (Poza, 2008):

a) Obtención y registro de señales
   i. Detección, muestreo, cuantificación y digitalización de la señal biomédica.
   ii. Preprocesado de la señal para la eliminación de espurios e interferencias de otros elementos internos y externos del mismo sistema biológico.
   iii. Almacenamiento y/o transmisión de la señal preprocesada.

b) Procesado
   i. Segmentación de la señal.
   ii. Filtrados y/o transformación de la señal.
   iii. Determinación de los patrones que son objeto de detección.

c) Clasificación
   i. Extracción de características.
   ii. Clasificación de la señal.

Este TFG se va a centrar en la fase de procesado. Aunque no es objeto de este proyecto actividades relacionadas con la obtención o clasificación de señales biomédicas, serán temas colaterales de interés. En concreto, nuestra tarea se va a centrar en la optimización de dos *scripts* desarrollados en MATLAB®. El primero de ellos es un método no lineal con alto coste computacional que acepta como entrada una señal de longitud variable y devuelve una pendiente negativa compuesta por 10 elementos. El segundo realiza procesado de matrices cuyos datos se han generado a través de métodos espectrales y que emplea un alto coste computacional a reducir.

2. INTRODUCCIÓN AL PROCESADO DE SEÑALES BIOMÉDICAS

3. COMPUTACIÓN PARALELA

3.1. FUNDAMENTOS

Las últimas generaciones de ordenadores permiten el almacenamiento de bases de datos de grandes dimensiones para su posterior análisis. No obstante, puede que la cantidad de datos a procesar sea tan ingente que el tiempo de computación requerido sea prohibitivamente alto (Vajteršic et al., 2009). En este contexto, nace el paradigma de la computación paralela: si varios procesadores cooperan entre sí para solucionar un problema de computación, entonces el tiempo requerido para completar la tarea podría reducirse de forma significativa. Pero la aceleración no es la única ventaja, al aumentar el número de núcleos del procesador, la calidad del resultado también mejora sin alterar el retardo (Vajteršic et al., 2009) .

En la Figura 1 puede verse la estructura de un ordenador secuencial. Un procesador ejecuta una secuencia de instrucciones de un controlador. Esa rutina se aplica sobre una secuencia de datos obtenida de la RAM. Se mete una entrada y se obtiene una salida.

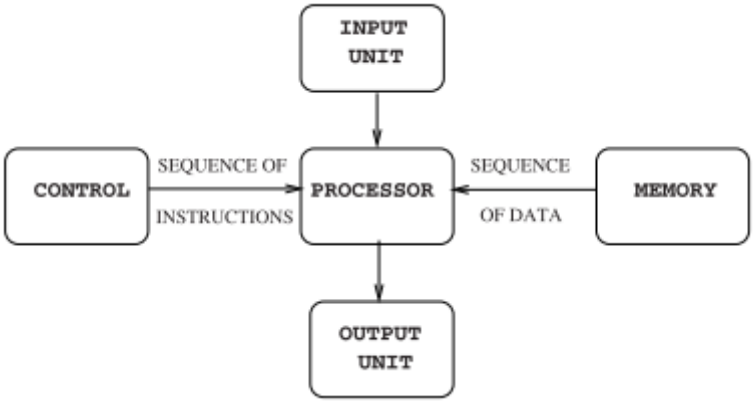

Figura 1. Esquema de un ordenador secuencial (Vajteršic et al., 2009).

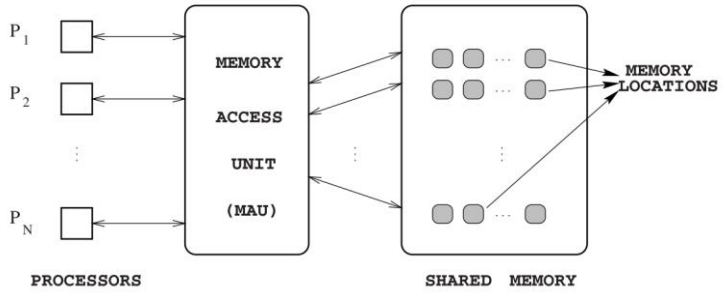

Figura 2. Modelo de memoria compartida (Vajteršic et al., 2009).

3. COMPUTACIÓN PARALELA

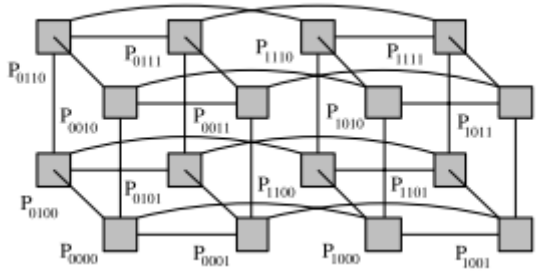

Figura 3. Modelo de red interconectada (Vajteršic et al., 2009).

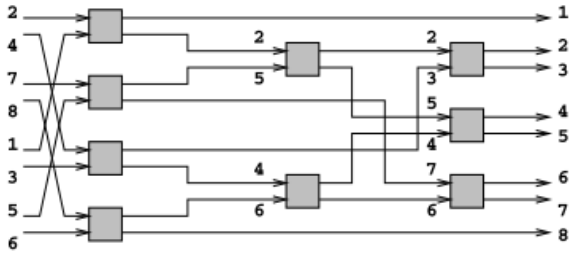

Figura 4. Modelo de circuitos (Vajteršic et al., 2009).

Sin embargo, en una estructura paralela, tenemos varios procesadores dispuestos de distintas formas, procesando cada uno la misma o distinta información. Podemos tener varias entradas y producir varias salidas. En función de ello, distinguimos distintos modelos de computación paralela (Vajteršic et al., 2009):

- Memoria compartida. Varios procesos acceden a una misma memoria para realizar de operaciones de lectura y escritura. (Figura 2). Una Unidad de Acceso a Memoria (MAU, *Memory Access Unit*) actúa como intermediaria entre los procesos y la memoria. Requiere temporización para cada procesador para que no se corrompa la memoria si acceden dos procesos o más a la vez.
- Red interconectada. Todos los procesadores se interconectan entre sí formando una red (Figura 3) y cada uno tiene su propia memoria local. Este diseño soluciona el inconveniente del cuello de botella de la MAU del caso anterior. Hay que decidir las conexiones existentes entre procesadores. No sería escalable establecer todos los enlaces posibles si el número de nodos es elevado. Por otro lado, si no hay los suficientes, puede ser excesivamente alto el diámetro de la red, es decir, el número mínimo de saltos entre los dos nodos más alejados entre sí.
- Circuitos. Este esquema se compone de varios circuitos, cada uno de ellos compuesto por distintos procesadores por los que va pasando. La salida de un nodo intermedio es la entrada al siguiente. El conjunto de uno o varios procesadores de distintos circuitos a la misma altura recibe el nombre de etapa. Los procesadores pertenecientes a una misma etapa pueden funcionar en paralelo. La Figura 4 propone una instancia de esta

arquitectura con nueve comparadores a modo de procesadores distribuidos a lo largo de 3 etapas.

Para hacer posible la computación paralela, otra de las tareas fundamentales consiste en la implementación de algoritmos fuertes y flexibles que coordinen el proceso de paralelización. Los cuatro pasos fundamentales en el desarrollo de un algoritmo de paralelización son (Vajteršic et al., 2009):

1. Particionado. Descomposición del problema en pequeñas partes y maximización del número de tareas que pueden ser ejecutadas a la vez.
2. Análisis de comunicación. Determinación de aquellos intercambios de información necesarios entre las distintas partes. Esto puede entenderse como un esquema de comunicaciones en el que los nodos son las tareas y los enlaces entre ellos, canales de conexión.
3. Control de granularidad. Intento de reducción de comunicaciones innecesarias entre tareas, mediante la aglomeración de las mismas en grupos diferentes, dando lugar a un menor número existente de ellas pero con más funcionalidades cada una.
4. Mapeo. Asignación de trabajos a los procesadores junto a la optimización del compromiso entre costes y grado de paralelismo.

3.2. COMPUTACIÓN PARALELA EN MATLAB®

En noviembre de 2004, MathWorks lanzó dos herramientas MATLAB® llamadas *Distributed Computing Toolbox* y MATLAB® *Distributed Computing Engine*, que más tarde pasarían a ser el *Parallel Computing Toolbox* y *Distributed Computing Server*, respectivamente (Sharma & Martin, 2009).

Parallel Computing Toolbox nos permite resolver programas con una gran carga computacional usando procesadores *multicore*, unidades de procesamiento gráfico (GPU, *Graphics Processing Unit*) y *clusters*. Dicho *toolbox* hace uso de la capacidad de procesado del *hardware* de nuestra computadora, ejecutando las aplicaciones en varios *workers* que funcionan de forma local. También se pueden ejecutar las aplicaciones, sin modificar el código, en un *cluster* o en un servicio de computación en malla (*grid computing*) (Mathworks, 2015; Ploskas, 2016). *Parallel Computing Toolbox* es capaz de mejorar el rendimiento en diversos tipos de situaciones (Ploskas, 2016):

- Aplicaciones con segmentos de código repetitivos y bucles. Cada iteración se evalúa por separado en un bucle paralelo con la única restricción de que dichas repeticiones han de ser independientes entre sí.
- Programas con una serie de tareas que no dependen unas de otras. También se puede implementar un bucle paralelo.
- Evaluación de un mismo código sobre distintos conjuntos de datos a la vez. Para ello, se utiliza un conjunto de *workers* que trabajan a la vez con el mismo código, pero con distintos datos.

3. COMPUTACIÓN PARALELA

- Información demasiado grande, de tal forma que no puede ser almacenada en la memoria de nuestro dispositivo y que, por tanto, ha de distribuirse entre distintos *workers* de tal forma que cada uno trabaje con una parte de los datos.
- Mejora de rendimiento si el código se ejecuta en paralelo o en una GPU.

El *Parallel Computing Toolbox* pone a nuestra disposición diferentes herramientas de paralelización en función de nuestras necesidades, ya sea para realizar un procesado más rápido o porque nuestros datos sean demasiado grandes como para entrar en la memoria. La Figura 5 muestra un organigrama de soluciones, muchas de las cuales se comentarán en el Capítulo 4 (Mathworks, 2015).

Distributed Computing Server de MATLAB® se compone de varios *workers* que reciben tareas de computación desde el lado del cliente a través de las funciones implementadas en el *Parallel Computing Toolbox* (Figura 6). En este TFG se utilizará la palabra *worker* para hacer referencia a los procesos que se ejecutan en un *cluster* del *Distributed Computing Server*.

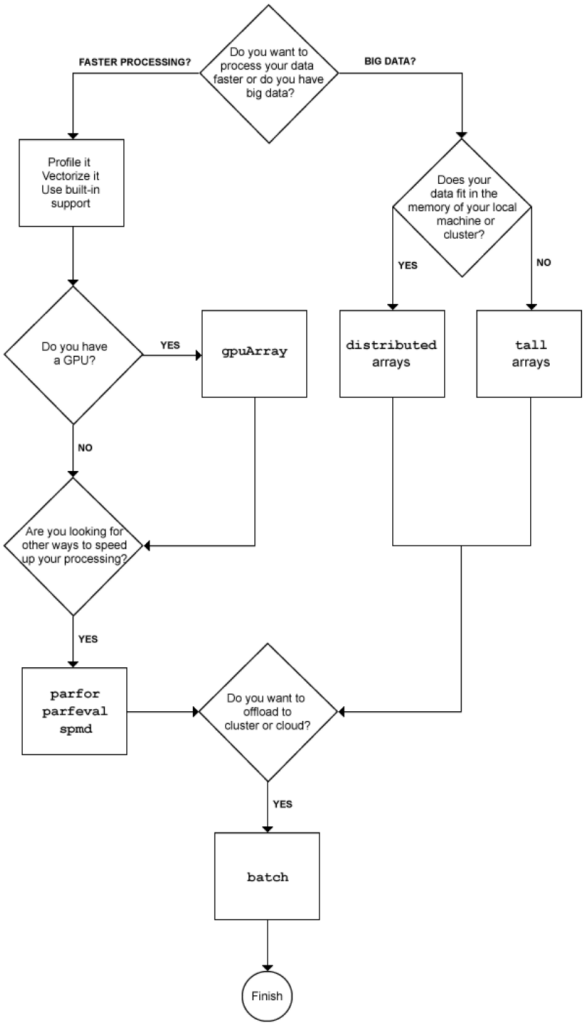

Figura 5. Organigrama para la elección de la mejor solución de paralelización (Mathworks, 2015).

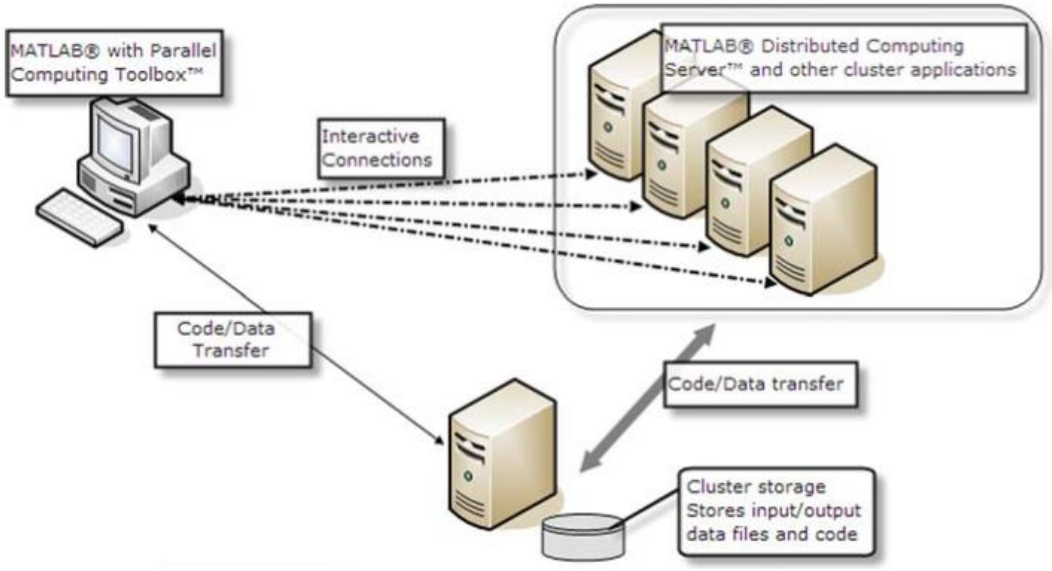

Figura 6 Montaje típico del cluster de Distributed Computing Server y su conexión con el cliente, donde está instalado el Parallel Computing Toolbox (Sharma & Martin, 2009)

3. COMPUTACIÓN PARALELA

4. HERRAMIENTAS Y TÉCNICAS DE PARALELIZACIÓN

En este capítulo se introducen las principales técnicas y herramientas de paralelización que podemos aplicar a nuestros programas en MATLAB®.

4.1. PROFILE

`Profile` es una herramienta que sirve para determinar en qué líneas del código el programa gasta más tiempo. Una vez que se han determinado dichas líneas, se podría evaluar su mejora de rendimiento con otras herramientas. Además, es útil como depurador, ya que si el programa lanza un error, se pueden ver qué líneas se han ejecutado y cuáles no. La Figura 7 muestra una captura de pantalla de esta aplicación (Mathworks, 2015).

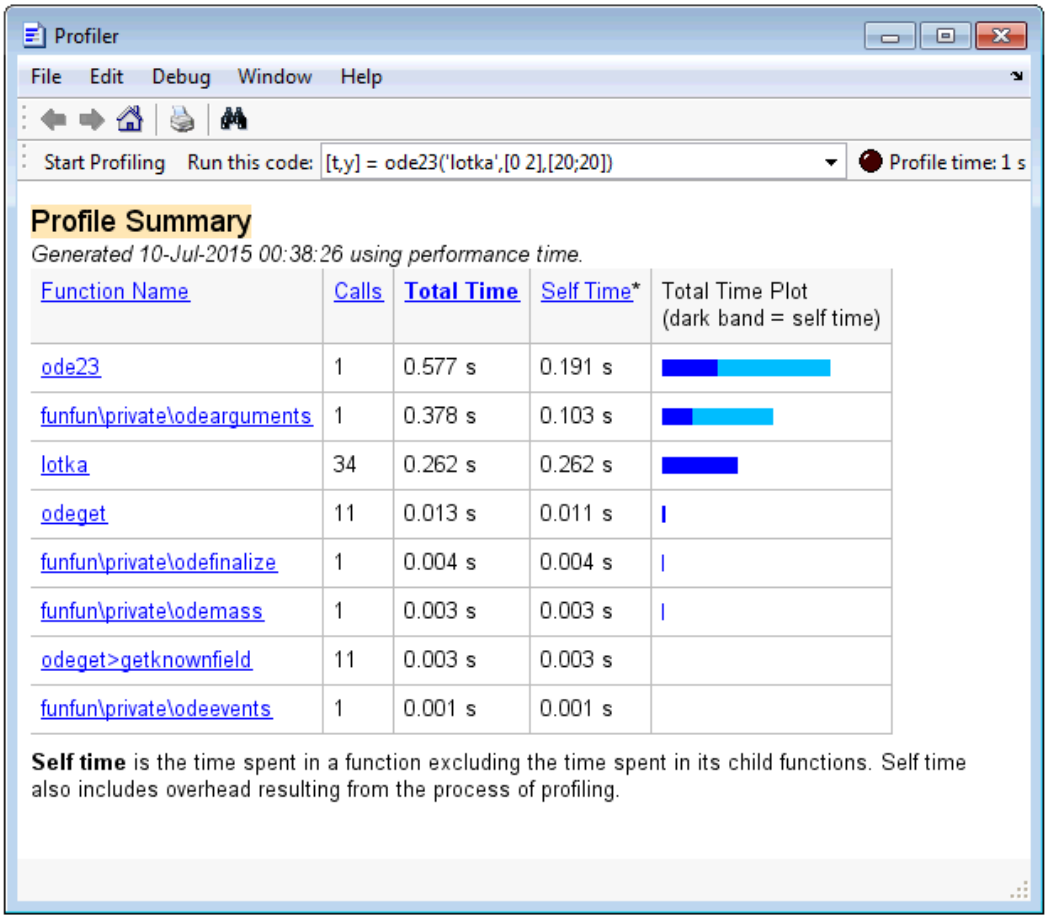

Figura 7. Herramienta "profile" (Mathworks, 2015).

4.2. VECTORIZACIÓN

La vectorización es una técnica en la que un código que opera sobre cada uno de los números escalares de un conjunto determinado mediante un bucle, se reformula de tal forma que opere directamente sobre esos conjuntos en forma de vectores o matrices (Mathworks, 2015).

El diseño de MATLAB® está optimizado para realizar operaciones con vectores y matrices. La vectorización es útil por varias causas (Mathworks, 2015):

- Código más inteligible. Facilidad de comprensión del cometido del programa.
- Menos probabilidad de fallo. Disminución de la probabilidad de error en el código como consecuencia de la reducción de las líneas a ejecutar.
- Mejora del rendimiento. Mayor rapidez en la ejecución del código respecto de su versión equivalente con bucles orientados a escalares.

Para poder vectorizar un código o parte de él, es necesario que las iteraciones del bucle correspondiente sean independientes entre sí. Sirva como ejemplo el de las Figuras 8 y 9, donde se observa la vectorización de un bucle que va asignando a un vector un valor en cada iteración y es transformado con una línea que asigna a todas las posiciones del vector a la vez sus valores (Mathworks, 2015).

4.3. PARFOR

Un bucle `parfor` en MATLAB® es similar a un bucle `for` ordinario. Sin embargo, sus iteraciones se ejecutan simultáneamente entre varios *workers*, que deberán haber sido creados dentro de una *pool* con carácter previo. El bucle `parfor` es iniciado en el cliente, que envía los datos necesarios a los *workers*, los cuáles se reparten las iteraciones del bucle y envían los resultados de vuelta al cliente. Para garantizar la independencia entre iteraciones existen ciertas restricciones a cumplir dentro de la cláusula `parfor.` La Figura 10 resume el funcionamiento de esta herramienta (Mathworks, 2015; Ploskas, 2016).

```
i = 0;
for t = 0:.01:10
    i = i + 1;
    y(i) = sin(t);
end
```

Figura 8. Versión sin vectorizar (Mathworks, 2015).

```
t = 0:.01:10;
y = sin(t);
```

Figura 9. Versión vectorizada (Mathworks, 2015).

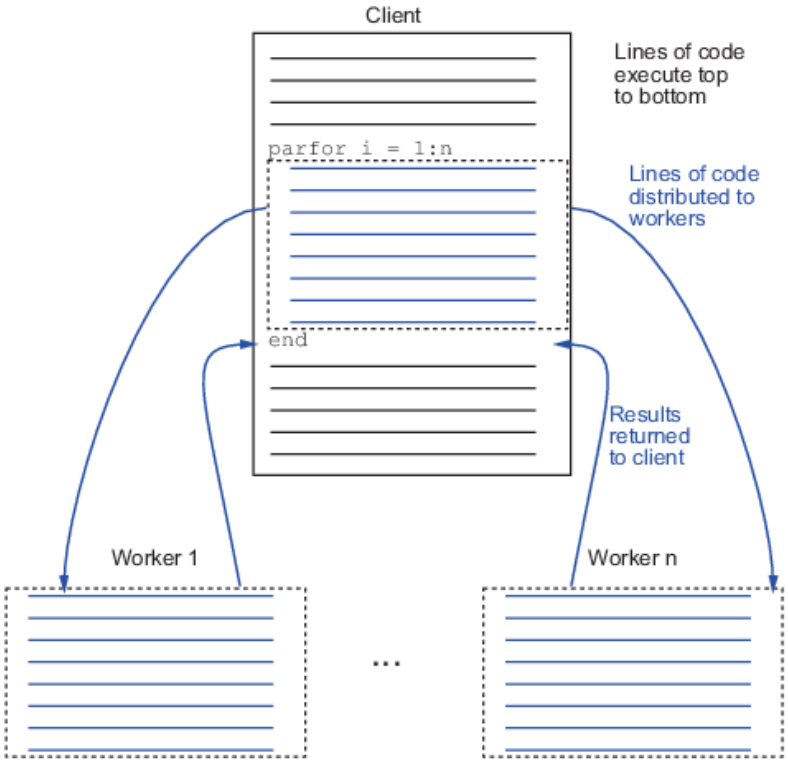

Figura 10. Funcionamiento de parfor *(Mathworks, 2015).*

4.4. PARFEVAL

Esta función permite la ejecución de una función determinada de forma asíncrona en un *worker* de MATLAB®, lo que permite que continúe la ejecución del programa principal sin que se bloquee. El método devuelve un objeto con los resultados, los cuales pueden ser recuperados con la función `fetchOutputs`.

4.5. SPMD

Esta cláusula permite paralelizar un fragmento de código entero. No se restringe solo a bucles, como `parfor`. Se ejecuta el mismo trozo de programa en todos los *workers,* cada uno de ellos trabajando con distintos datos. Los fragmentos previo y posterior de la cláusula `spmd` se ejecutan en el cliente, mientras que el contenido de la misma se ejecuta en los *workers.*

Al igual que en casos anteriores, también requiere la creación de una *pool* antes de poder utilizar esta herramienta. Los *workers* pueden comunicarse entre sí usando un determinado número de funciones. Se utilizan vectores `distributed` y `codistributed` para particionar grandes conjuntos de datos (Ploskas, 2016).

4.6. PROCESADO EN GPU

Si después de haber utilizado las herramientas anteriores, nuestro código sigue siendo demasiado lento o no tan rápido como deseáramos, se puede intentar utilizar una GPU de nuestro ordenador para intentar realizar los cálculos a mayor velocidad. Si todas nuestras funciones son soportadas por la GPU, solo tenemos que transferir los datos iniciales a la misma y traer de vuelta los datos finales. Hay que tener en cuenta que puede añadir un poco de sobrecarga y acabar aumentando el tiempo de ejecución si se transfiere demasiada información. Asimismo, hemos de estimar la memoria que vamos a necesitar, ya que igualmente podría desbordarse la memoria en pleno proceso (Mathworks, 2015).

4.7. CLUSTERS

El *Parallel Computing Toolbox* y el MATLAB® *Distributed Computing Server* nos permiten resolver tareas en computadoras multiprocesador. Varios grupos de tareas se agrupan en *jobs*, que han de definirse en el cliente. En los *clusters* ha de estar instalado el otro *toolbox* de computación distribuida. Para llevar a cabo una coordinación adecuada entre trabajos y tareas, MATLAB® cuenta con su propio *scheduler* de trabajos (MJS, MATLAB® *Job Scheduler*), que reenvía los datos de entrada del cliente a los *workers* y hace lo propio con los resultados de los *workers* al cliente.

En la Figura 11 se puede observar un ejemplo sencillo de esta arquitectura.

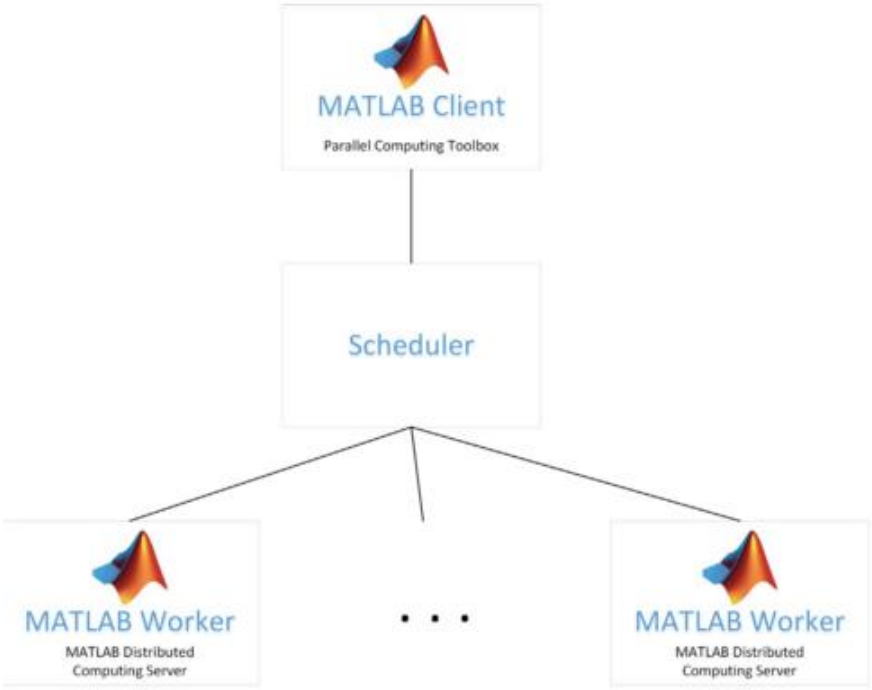

Figura 11. Esquema de una red con un cliente, un scheduler y varios workers de MATLAB® (Mathworks, 2015).

5. RESULTADOS

5.1. ESTRATEGIAS DE OPTIMIZACIÓN

Durante el desarrollo del TFG, se han identificado varios casos en los que puede ser posible aplicar técnicas de paralelización mediante MATLAB®. En este capítulo se listan dichas situaciones, las técnicas aplicadas y los resultados obtenidos.

Todos los tiempos se han medido como la media aritmética del retardo de 5 ejecuciones en condiciones idénticas. Aunque no todas las medidas presentadas en este capítulo se han tomado desde un mismo equipo, las medidas están tomadas con una sola computadora dentro de una misma estrategia, y además, se ha tomado como hipótesis nula que la mejora relativa de rendimiento entre dos versiones del código es siempre la misma, siempre y cuando las medidas se tomen desde un mismo dispositivo. Los equipos utilizados se referenciarán como Ordenador 1, Ordenador 2 y Ordenador 3.

Las especificaciones técnicas de cada una de estas 3 máquinas son las siguientes:

- **Ordenador 1:** Ordenador portátil con Intel® Core(TM) i7-6700HQ CPU @ 2.60 GHz 2.60 GHz, RAM de 8 GB y 4 *cores.*
- **Ordenador 2:** Ordenador de sobremesa con Intel® Core(TM) i7-3930K CPU @ 3.20 GHz 3.20 GHz, RAM de 32 GB y 6 *cores.*
- **Ordenador 3:** Ordenador de sobremesa con Intel® Xeon E5-1620v4 CPU @ 3.50 GHz 3.50 GHz, RAM de 8 GB y 4 *cores.*

Las 7 primeras estrategias se han utilizado para intentar paralelizar un programa utilizado por el GIB de la UVA llamado *nonlinear_forecasting_serie*, que tiene como entrada una trama de magnetoencefalograma de longitud variable procedente de un único canal. Como salida devuelve la pendiente de los valores del coeficiente de correlación entre los valores predichos y los reales. Se pueden utilizar 3 métodos alternativos: Aftanas, Blinowska y Yunfan. El lector interesado puede consultar (Aftanas et al., 1997; Blinowska et al., 1991; Yunfan et al., 1998) respectivamente, ya que los detalles del algoritmo no son objeto de estudio de este TFG. Por simplicidad se trabajará solamente con el método Aftanas, al ser los 3 muy similares.

La última estrategia se utilizará para intentar optimizar un programa que realiza una transformación tiempo-frecuencia mediante una transformada *wavelet* continua (CWT, *Continuous Wavelet Transform*). Toma como entrada una matriz con varias muestras temporales por cada uno de los canales en los que se ha registrado la actividad cerebral y devuelve, entre otras salidas, la densidad espectral de potencia (PSD, *Power Spectral Density*) con 139 *trials*, cada uno de ellos con información de un número variable de canales a lo largo de 500 muestras temporales y 108 muestras frecuenciales.

Por simplicidad, a partir de ahora, nos referiremos al primer programa como *Programa NLF* y al segundo, como *Programa CWT*.

5. RESULTADOS

Estrategia de Vectorización: Se ha hecho uso de la herramienta `profile` para averiguar qué líneas son las que más sobrecargan el *Programa NLF*. Se ha observado que, para una entrada de 10000 muestras y una medición en el Ordenador 1, una de esas líneas tarda más de 4 minutos en ejecutarse y hay otras dos líneas más abajo que tardan más de 12 y 10 segundos respectivamente, aunque estos valores pueden variar de ejecución en ejecución según el número de procesos ejecutándose y el tamaño de trama (Figura 12).

Los tiempos de ejecución para valores entre 1000 y 10000 muestras se indican en la Tabla 1 y en la Figura 13. Se ha utilizado el Ordenador 1 para obtenerlas.

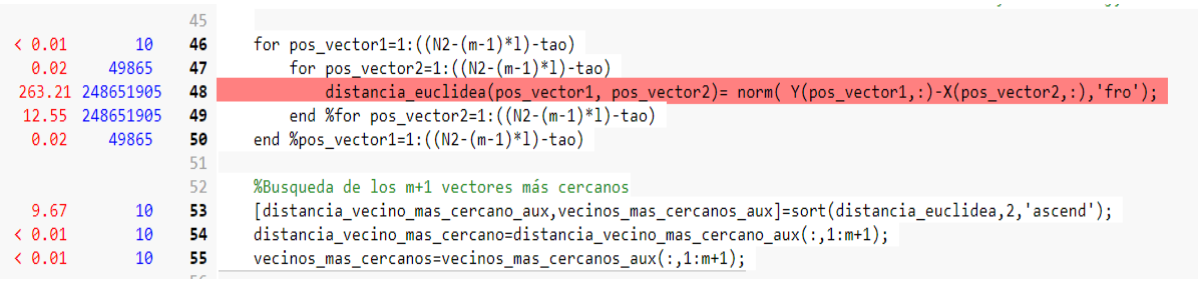

Figura 12. Profile de la versión inicial del Programa NLF.

Tabla 1. Tiempos de ejecución de la versión original del Programa NLF.

Número de muestras	Tiempo (s)
1000	2,87
2000	10,68
3000	25,49
4000	34,21
5000	54,27
6000	74,11
7000	105,03
8000	140,59
9000	180,49
10000	238,83

5. RESULTADOS

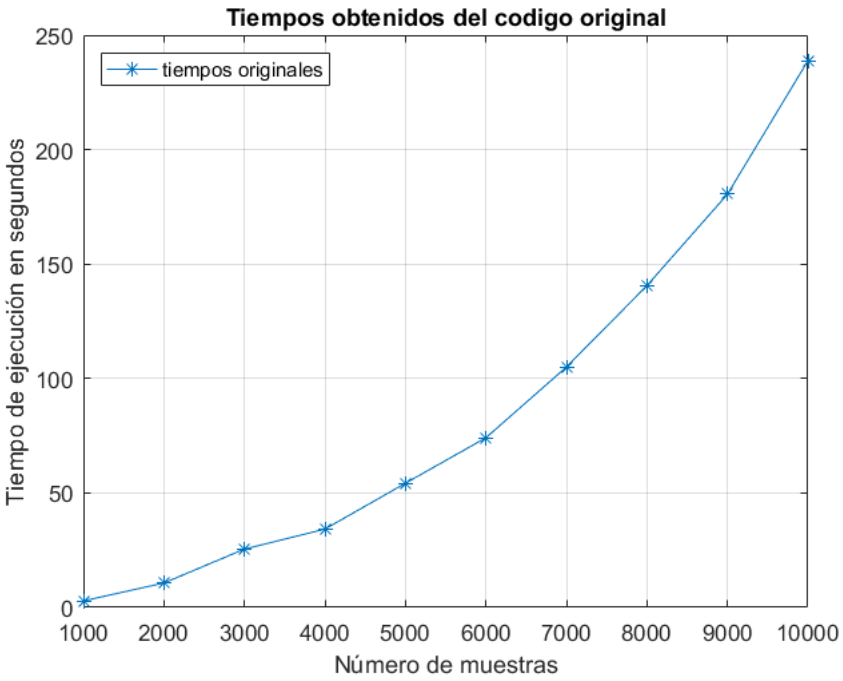

Figura 13. Gráfica de tiempos de la versión original del Programa NLF.

Se ha decidido hacer uso de la vectorización para intentar aliviar la carga de este programa. En concreto, se ha empleado la función `vecnorm` de MATLAB® que es capaz de calcular el módulo de varias filas a la vez de una misma matriz, por lo que el bucle `for` interno desaparecería.

Estrategia de Bucle Paralelo: Al mismo fragmento de código anterior, esta vez se le ha aplicado un bucle paralelo `for`. Más concretamente, el bucle `for` más externo se ha convertido en uno `parfor`. Se han tomado medidas con *pools* de 4 y 6 *workers* (Figura 14).

Estrategia de GPU: En esta ocasión se han intentado mejorar los resultados del caso anterior, ejecutando el *Programa NLF* en una tarjeta GPU NVIDIA Titan XP con arquitectura de GPU Pascal, 12 GB G5X de memoria de vídeo, 11.4 Gbps de frecuencia en la memoria y 1582 MHz de frecuencia acelerada.

Estrategia de GPU y vectorización: En pocas palabras, se ha intentado fusionar las Estrategias de GPU y de Vectorización.

Estrategia de `spmd`: Aquí se ha intentado paralelizar el fragmento de código anterior utilizando cláusulas `spmd` (Figura 15).

Estrategia de `spmd` y Vectorización: Ahora se ha intentado mezclar las Estrategias de Vectorización y de `spmd`, como se puede observar en la Figura 16.

5. RESULTADOS

```
parfor pos_vector1=1:((N2-(m-1)*l)-tao)
    v=zeros(1,(N2-(m-1)*l)-tao);
      for pos_vector2=1:((N2-(m-1)*l)-tao)
        v(pos_vector2)= norm( Y(pos_vector1,:)-X(pos_vector2,:),'fro');
    end %for pos_vector2=1:((N2-(m-1)*l)-tao)
    distancia_euclidea(pos_vector1,:)=v;
end %pos_vector1=1:((N2-(m-1)*l)-tao)
```

Figura 14. Bucle paralelo para la Estrategia de Vectorización.

```
X=zeros(N2-(m-1)*l,m,'distributed'); %Matriz que contiene en cada fila a un punto en el espacio m-dimensional
Y=zeros(N2-(m-1)*l,m,'distributed'); %Matriz que contiene en cada fila a un punto en el espacio m-dimensional



spmd(4)
    for i=1:N2-(m-1)*l
        X(i,:)=datos_library(i+(0:(m-1))*l); %Vectores m-dimensionales
        Y(i,:)=datos_prediction(i+(0:(m-1))*l); %Vectores m-dimensionales
    end %for i=1:N2-(m-1)*l
end


%Calculo de las predicciones
for tao=1:10 %Tiempo de prediccion

    %Calculo de las distancias euclideas.
    distancia_euclidea=zeros(((N2-(m-1)*l)-tao),((N2-(m-1)*l)-tao),'distributed');%Tabla que en la fila i y co
% spmd (4)                                                                    %irá almacenando las distancias
%  for pos_vector1=1:((N2-(m-1)*l)-tao)
%             Z=Y(pos_vector1,:)-X;
%             distancia_euclidea(pos_vector1,:)=vecnorm(Z(1:((N2-(m-1)*l)-tao),:),2);%PABLO: vecnorm es una fun
%  end                                                                       %entre el vector i y el vector j, ini
% end
    spmd(4)
        for pos_vector1=1:((N2-(m-1)*l)-tao)
            for pos_vector2=1:((N2-(m-1)*l)-tao)
                distancia_euclidea(pos_vector1, pos_vector2)= norm( Y(pos_vector1,:)-X(pos_vector2,:),'fro');
            end %for pos_vector2=1:((N2-(m-1)*l)-tao)
        end %pos_vector1=1:((N2-(m-1)*l)-tao)
    end


    dist_eucl=gather(distancia_euclidea);
    x=gather(X);
    y=gather(Y);
```

Figura 15. Cláusulas spmd *en el código, correspondientes a la* Estrategia de *spmd.*

```
X=zeros(N2-(m-1)*l,m,'distributed'); %Matriz que contiene en cada fila a un punto
Y=zeros(N2-(m-1)*l,m,'distributed'); %Matriz que contiene en cada fila a un punto



spmd(4)
    for i=1:N2-(m-1)*l
        X(i,:)=datos_library(i+(0:(m-1))*l); %Vectores m-dimensionales
        Y(i,:)=datos_prediction(i+(0:(m-1))*l); %Vectores m-dimensionales
    end %for i=1:N2-(m-1)*l
end


%Calculo de las predicciones
for tao=1:10 %Tiempo de prediccion

    %Calculo de las distancias euclideas.
    distancia_euclidea=zeros(((N2-(m-1)*l)-tao),((N2-(m-1)*l)-tao),'distributed');
spmd (4)                                                                    %irá a
 for pos_vector1=1:((N2-(m-1)*l)-tao)
           Z=Y(pos_vector1,:)-X;
           distancia_euclidea(pos_vector1,:)=vecnorm(Z(1:((N2-(m-1)*l)-tao),:),2);
 end                                                                      %entre el v
end
% spmd(4)
%     for pos_vector1=1:((N2-(m-1)*l)-tao)
%         for pos_vector2=1:((N2-(m-1)*l)-tao)
%             distancia_euclidea(pos_vector1, pos_vector2)= norm( Y(pos_vector1,:)
%         end %for pos_vector2=1:((N2-(m-1)*l)-tao)
%     end %pos_vector1=1:((N2-(m-1)*l)-tao)
% end


    dist_eucl=gather(distancia_euclidea);
    x=gather(X);
    y=gather(Y);
```

Figura 16. Código vectorizado y paralelizado con spmd*, correspondiente a la* Estrategia de *spmd* y Vectorización.

```
for pos_vector1=1:((N2-(m-1)*l)-tao)
    for pos_vector2=1:((N2-(m-1)*l)-tao)
        distancia_euclidea(pos_vector1, pos_vector2)= norm( Y(pos_vector1,:)-X(pos_vector2,:),'fro');
    end %for pos_vector2=1:((N2-(m-1)*l)-tao)
end %pos_vector1=1:((N2-(m-1)*l)-tao)

%Busqueda de los m+1 vectores más cercanos
f=parfeval(gcp, @sort,2,distancia_euclidea,2,'ascend');
[distancia_vecino_mas_cercano_aux,vecinos_mas_cercanos_aux]=fetchOutputs(f);
distancia_vecino_mas_cercano=distancia_vecino_mas_cercano_aux(:,1:m+1);
vecinos_mas_cercanos=vecinos_mas_cercanos_aux(:,1:m+1);
```

Figura 17. Programa NLF *modificado con la función* parfeval*, correspondiente a la* Estrategia de *parfeval*.

Estrategia de `parfeval`: En esta ocasión se ha evaluado el uso de la función `parfeval`, que sirve para ejecutar la llamada a una función de forma asíncrona en un *worker*. En este caso el método invocado es `sort`, ya que se ha detectado que sobrecarga mucho al tener que ordenar cientos de datos. La Figura 17 muestra el código modificado para que se ejecute mediante `parfeval`.

Estrategia de Optimización Aplicada al *Programa CWT*: Aquí ya hemos cambiado a optimizar nuestro segundo código, el *Programa CWT*. Se han obtenido los siguientes resultados antes de optimizar. Se han contemplado los casos para un número de 17, 32, 64 y 128 canales (Tabla 2).

Se han detectado tres regiones del código optimizables con la herramienta `profile`, que se exponen a continuación:

- De dos bucles `for` anidados que invocan a una función para calcular la transformada *wavelet*, realizar la conversión del `for` más externo en `parfor` (Figura 18 y Figura 19).
- Vectorización y paralelización del bucle que realiza el filtrado de los datos electroencefalográficos obtenidos (Figura 20 y Figura 21).
- Granularización de la función `save` que almacena en ficheros las distintas variables de salida del *Programa CWT*. Una de las salidas a almacenar es la PSD, que resulta ser la más pesada de todas con diferencia, al ocupar casi 2 GB de memoria. La función `save` guarda todas las variables con compresión, lo que aumenta el tiempo de ejecución. Se ha optado por guardar sin comprensión todas las variables cuyo espacio en memoria es del orden de los KB y guardar comprimida únicamente la PSD. Se ha decidido también almacenar la PSD por secciones en vez de en un único fichero, para reducir el coste computacional de la función `save`. La variable de barrido elegida para seccionar la PSD, por conveniencia, han sido los distintos valores de frecuencia establecidos. Esta granularización se ha insertado en un bucle `parfor` para optimizar el retardo computacional (Figura 22 y Figura 23). Debido a las restricciones de `parfor`, ha habido que invocar a la función `save` a través de una función dentro del bucle paralelo (Figura 24).

Tabla 2. Tiempos en segundos al ejecutar el Programa CWT *en su versión inicial.*

Número de canales	Tiempo medio (s)
17	160,35
32	230,55
64	547,31
128	1897,91

5. RESULTADOS

```
for epochs = 1:size(datos.data,1),
    % Se inicializa la estructura que contendrá la CWT de cada época
    cwtepoch = [];
    for sensor = 1:size(datos.data,2),
        cwtsensor = [];
        cwtsensor = cwt(squeeze(datos.data(epochs,sensor,:)), escalas, wavelet);
        % En lugar de ordenar por escalas, ordenamos por frecuencias
        cwtsensor = flipud(cwtsensor);
        % Se añade (en la 2ª dimension o una columna adicional) el bloque de CWTs para cada sensor
        cwtepoch = cat(2,cwtepoch,permute(cwtsensor,[2 3 1]));
    end % Fin del 'for' que recorre cada sensor
    % Se añade (en la 4ª dimension) el bloque de CWTs de cada época
    data(:,:,:,epochs) = cwtepoch;
end % Fin del 'for' que recorre las épocas
```

Figura 18. Sección para calcular la transformada wavelet sin paralelizar.

```
parfor epochs = 1:N %PABLO Veáse profile V2 15/03/2018 Se añade un parfor, que r
    % Se inicializa la estructura que contendrá la CWT de cada época
    cwtepoch = [];
    for sensor = 1:M
        %cwtsensor = [];
        cwtsensor = cwt(squeeze(datos.data(epochs,sensor,:)), escalas, wavelet);
        %F=parfeval(gcp,@cwt,1,squeeze(datos.data(epochs,sensor,:)), escalas, wa
        %cwtsensor=fetchOutputs(F);
        %cwtsensor = cwt(squeeze(datos.data(epochs,sensor,:)), wavelet);
        % En lugar de ordenar por escalas, ordenamos por frecuencias
        cwtsensor = flipud(cwtsensor);
        % Se añade (en la 2ª dimension o una columna adicional) el bloque de CWT
        cwtepoch = cat(2,cwtepoch,permute(cwtsensor,[2 3 1]));
    end % Fin del 'for' que recorre cada sensor
    % Se añade (en la 4ª dimension) el bloque de CWTs de cada época
    data(:,:,:,epochs) = cwtepoch;
end % Fin del 'for' que recorre las épocas
```

Figura 19. Sección para la calcular la transformada wavelet paralelizada.

```
for n_canal=1:size(eeg,2)
    matriz_sujeto_canal = eeg(:,n_canal);

    matriz_sujeto_canal = matriz_sujeto_canal - mean(matriz_sujeto_canal);
    eeg(:,n_canal) = filtfilt(numerador,denominador,matriz_sujeto_canal);
end
```

Figura 20. Bucle de la parte de filtrado sin paralelizar.

```
    eeg=eeg-mean(eeg);
    canales=size(eeg,2);
parfor n_canal=1:canales
    eeg(:,n_canal) = filtfilt(numerador,denominador,eeg(:,n_canal));
end

matriz_sujeto = eeg;
```

Figura 21. Bucle de la parte de filtrado paralelizado.

5. RESULTADOS

```
try
save([dir_ppal '/CWT17 faltan/' registros(nreg).name(1:5)],'CWTData','-v7.3');
catch me
save([dir_ppal '/CWT17 faltan/' registros(nreg).name(1:5)],'CWTData','-v7.3');
end
```

Figura 22. Sección del código del salvado de datos sin paralelizar.

```
try
   save([dir_ppal '\CWT17\' registros(nreg).name(1:5)],'CWTData','-v7.3','-nocompression');%PABL(
save([dir_ppal '\CWT17\' registros(nreg).name(1:5)],'Cconfig','-v7.3','-nocompression');
N=size(Cpsd,4);
psdsliced=zeros(size(Cpsd,1),size(Cpsd,2),size(Cpsd,3));
%varcell={};
  for i=1:N
     varcell=cat(1,varcell,['psdsliced' num2str(i)]);
  end
parfor i=1:N
    psdsliced=Cpsd(:,:,:,i);
    currentfile=sprintf('psd%d.mat',i);
    %save([dir_ppal '\CWT17\' registros(nreg).name(1:5)],'Cpsliced','-v7.3');
    parsave(dir_ppal, currentfile ,psdsliced);
end
 %save([dir_ppal '\CWT17\' Psd],'Cpsd','-v7.3','-nocompression');
save([dir_ppal '\CWT17\' registros(nreg).name(1:5)],'-append','Cscales','-nocompression');
save([dir_ppal '\CWT17\' registros(nreg).name(1:5)],'-append','Cfrequency','-nocompression');
save([dir_ppal '\CWT17\' registros(nreg).name(1:5)],'-append','Cresolucion_t','-nocompression');
save([dir_ppal '\CWT17\' registros(nreg).name(1:5)],'-append','Cresolucion_f','-nocompression');
 save([dir_ppal '\CWT17\' registros(nreg).name(1:5)],'-append','Cname','-nocompression');
 %parsave(dir_ppal,registros(nreg).name(1:5),CWTData);
  puede que suponga un sacrificio de más memoria, (de 1,7 GB a 2 GB) pero este cambio, junto con
  hacen que el tiempo de ejecución se haya reducido en un 50% respecto
  de la versión original
catch me
  save([dir_ppal '\CWT17\' registros(nreg).name(1:5)],'CWTData','-v7.3');
  %parsave(dir_ppal,registros(nreg).name(1:5),CWTData);

end
```

Figura 23. Sección del código de salvado de datos paralelizada.

```
function parsave(dir,registro,Data)
    save([dir '\CWT17\' registro],'Data','-v7.3');
end
```

Figura 24. Definición de la función auxiliar para poder salvar datos estando dentro de un bucle parfor.

Si lo miramos en la misma herramienta `profile`, observamos como algunas líneas tardan más de lo normal. En la Figura 25 podemos ver que la línea que invoca al cálculo de la CWT tarda un minuto y la línea que guarda la PSD junto con el resto de los datos ocupa más de un minuto y medio del retardo total de ejecución. En la Figura 26, vemos que la línea que invoca al filtrado de datos tarda 4 segundos. Estas medidas han sido tomadas en el Ordenador 3.

5.2. RESULTADOS DE RENDIMIENTO

Para la *Estrategia de Vectorización*, tras haber vectorizado, observamos que para una entrada de 10000 muestras la misma línea ya no sobrecarga tanto, ya que apenas necesita 10 segundos para ejecutarse (Figura 27). También hemos observado que con 10000 muestras se ha pasado de tardar 4 minutos a tardar tan solo 20 segundos en ejecutar el programa. Si prestamos atención a las gráficas (Figura 28), podemos observar que la pendiente de la curva de tiempos se reduce mucho y pasa a tener un comportamiento casi lineal como resultado de la vectorización. Se puede apreciar que el rendimiento es aproximadamente del 90% (Tabla 3) para cualquier número de muestras. Los tiempos han sido medidos en el Ordenador 1.

Para la *Estrategia de Bucle Paralelo*, se observa que para un número bajo de muestras, como 1000, el tiempo de ejecución se reduce en torno al 50% (ver Tabla 4). Cuando empieza a aumentar el número de muestras, el rendimiento aumenta en función del número de *workers* que contenga nuestra *pool* creada. Si son 4 *workers*, el rendimiento obtenido oscila en torno al 70%, y si es de 6, en torno al 76%. Puede apreciarse en la Figura 29 que todas las curvas presentan comportamientos exponenciales, aunque la mayor pendiente la presenta la curva de tiempos originales, mientras que las otras, tienen mucha menos pendiente y tienden a un comportamiento lineal. Los tiempos han sido medidos en el Ordenador 2 (Tabla 4 y Figura 29).

Para la *Estrategia de GPU*, se ha realizado una única medición en el Ordenador 3 para una entrada de tamaño 1000 muestras, y se ha obtenido un tiempo de 481 segundos, lo que nos lleva a descartar inmediatamente esta aproximación por ineficiente.

En la *Estrategia GPU y Vectorización*, podemos observar que el uso de la GPU sigue ralentizando el procesado del *Programa NLF* en un porcentaje inversamente proporcional a la longitud de la señal de entrada (ver Tabla 5). En la Figura 30, se puede observar que la combinación de las técnicas de GPU y vectorizado convierten al programa en un método lineal mucho más eficiente que en su versión original, aunque con el vectorizado por sí solo se siguen obteniendo resultados algo más satisfactorios. Los resultados se han obtenido del Ordenador 3 (Tabla 5 y Figura 30).

5. RESULTADOS

```
63.79       1   246     CWTData =  CalcularCWT_NST(cleanEstimulo,pseudo_escala,wav_madre);
                247     % Se guarda la seï¿¼al transformada
< 0.01      1   248     try
96.92       1   249     save([dir_ppal '\CWT17\' registros(nreg).name(1:5)],'CWTData','-v7.3');
                250     catch me
                251     save([dir_ppal '\CWT17\' registros(nreg).name(1:5)],'CWTData','-v7.3');
                252     end
```

Figura 25. Profile del Programa CWT *(I).*

```
 4.12       1    95     matriz_sujeto = FiltrarDatos_NST(filtro_banda,dir_capado,registros(nreg).name(1:5));
```

Figura 26. Profile del Programa CWT *(II).*

```
< 0.01      1    45 for tao=1:10 %Tiempo de prediccion
                 46
                 47     %Calculo de las distancias euclideas.
  0.22     10    48     distancia_euclidea=zeros(((N2-(m-1)*l)-tao),((N2-(m-1)*l)-tao));%Tabla que en
                 49                                                                     %irá almacenan
< 0.01     10    50  for pos_vector1=1:((N2-(m-1)*l)-tao)
  1.33  49865    51            Z=Y(pos_vector1,:)-X;
 10.65  49865    52            distancia_euclidea(pos_vector1,:)=vecnorm(Z(1:((N2-(m-1)*l)-tao),:),2);
< 0.01  49865    53  end                                                                %entre el v
```

Figura 27. Profile del Programa NLF *tras haber aplicado la* Estrategia de Vectorización.

Tabla 3. Resultados tras haber aplicado la Estrategia de Vectorización.

Numero de muestras	**Tiempo original (s)**	**Tiempo de ejecución tras aplicar la optimización de la *Estrategia de Vectorización* (s)**	**Mejora de rendimiento (%)**
1000	2,87	0,34	88,15
2000	10,68	1,08	89,88
3000	25,49	2,23	91,25
4000	34,21	3,78	88,95
5000	54,27	5,94	89,05
6000	74,11	9,37	87,36
7000	105,03	11,2	89,34
8000	140,59	14,29	89,84
9000	180,49	19,00	89,47
10000	238,83	24,72	89,65

5. RESULTADOS

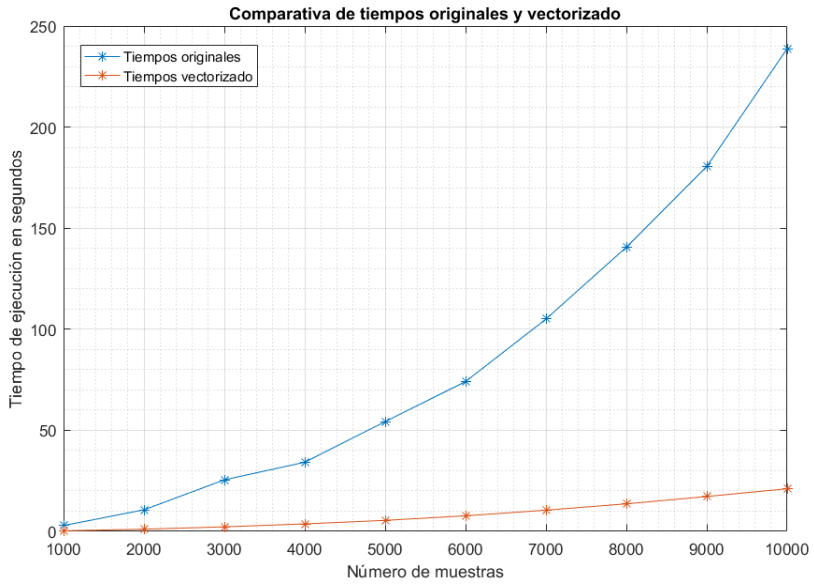

Figura 28. Gráfica de tiempos tras haber aplicado la Estrategia de Vectorización.

Tabla 4. Comparativa de los tiempos de ejecución del código original y el código con parfor_4, *con* parfor_6 *y con vectorizado.*

Número de muestras	**Tiempo original (s)**	**Tiempo de ejecución tras aplicar la optimización de la *Estrategia de Bucle Paralelo* (s)**					
		***Parfor_4* (s)**	**Mejora de rendimiento *parfor_4* (%)**	***Parfor_6* (s)**	**Mejora de rendimiento *parfor_6* (%)**	**Vectorizado (s)**	**Mejora de rendimiento vectorizado (%)**
1000	2,28	1,37	39,91	1,37	39,91	0,29	87,28
2000	9,13	3,19	65,06	2,76	69,77	1,08	88,17
3000	20,84	6,30	69,77	5,16	75,24	2,24	89,25
4000	36,98	10,77	70,88	8,61	76,72	3,71	89,97
5000	58,40	16,63	71,52	12,99	77,76	5,45	90,67
6000	84,13	23,89	71,60	18,85	77,59	7,70	90,85
7000	114,48	32,78	71,37	25,50	77,73	10,49	90,84
8000	150,02	42,96	71,36	33,11	77,93	13,62	90,92
9000	190,66	54,38	71,48	41,47	78,25	17,25	90,95
10000	234,38	66,70	71,54	51,08	78,21	21,08	91,01

5. RESULTADOS

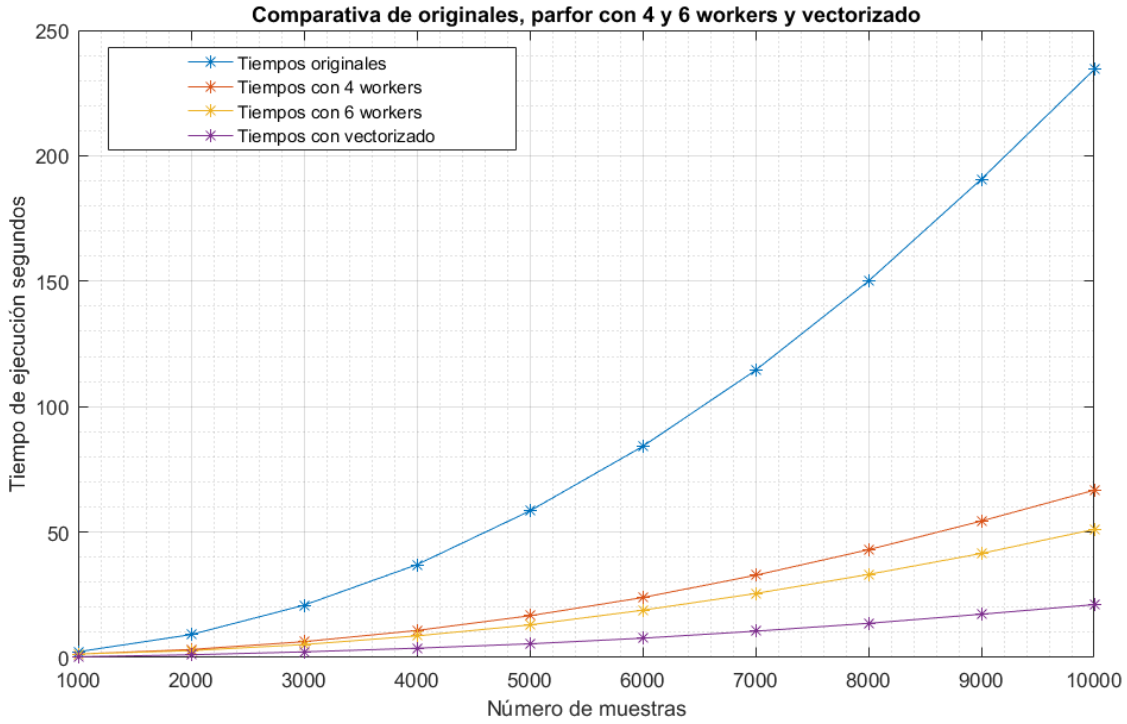

Figura 29. Comparativa de vectorizado, parfor_4 y parfor_6 *(*Estrategia de Bucle Paralelo*).*

Tabla 5. Resultados al juntar las Estrategias de GPU y de vectorización.

Número de muestras	**Original (s)**	**Tiempo de ejecución tras aplicar la optimización de la *Estrategia de GPU y Vectorización* (s)**		
		Vectorizado (s)	**Vectorizado y GPU (s)**	**Mejora de rendimiento respecto del vectorizado (%)**
1000	1,93	0,23	5,77	-2424,85
2000	7,87	0,75	11,66	-1461,08
3000	17,63	1,72	17,86	-940,49
4000	31,89	2,83	23,99	-747,58
5000	50,66	4,45	29,82	-570,20
6000	74,28	6,38	36,12	-466,42
7000	99,41	8,95	42,88	-379,33
8000	129,74	11,23	49,37	-339,55
9000	164,76	14,21	56,23	-295,79
10000	203,78	17,78	63,34	-256,20

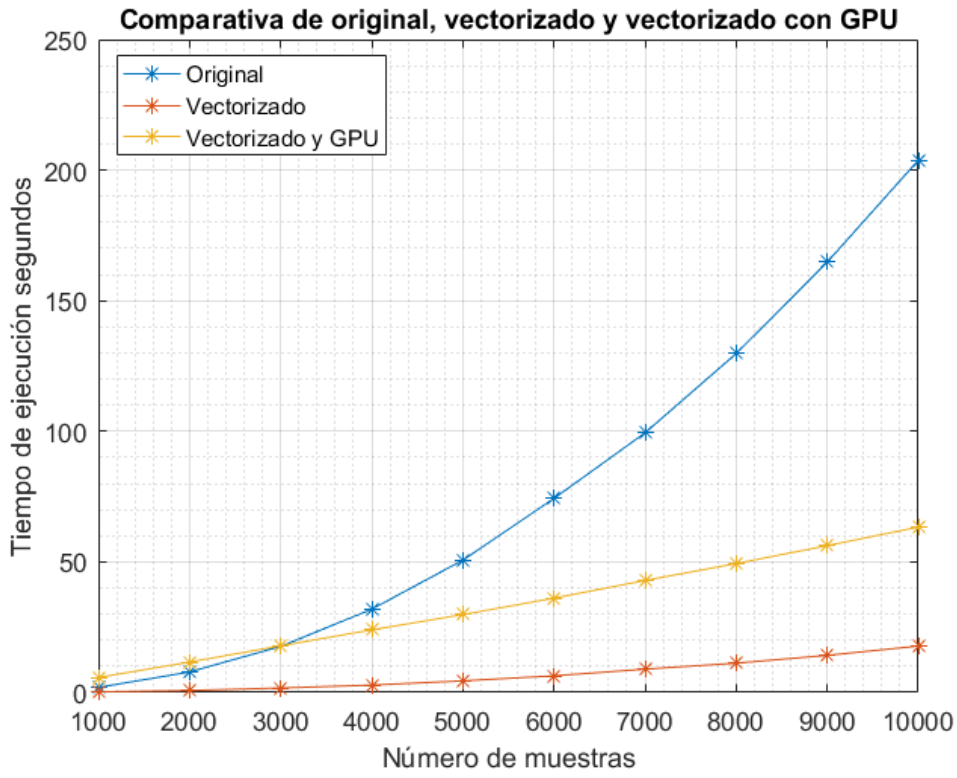

Figura 30. Gráfica comparativa para los tiempos de vectorizado a secas y vectorizado con GPU.

En la *Estrategia de* `spmd` se ha realizado una única medición en el Ordenador 1 para un tamaño de entrada de 100 muestras, obteniendo un tiempo de 153 segundos, lo que nos sugiere que esta técnica no es eficiente.

En la *Estrategia* `spmd` *y Vectorización*, para la curva de vectorizado y `spmd`, apreciamos un tramo de tiempo constante hasta las 3000 muestras, que creemos que se debe a la sobrecarga introducida cuando el tamaño de entrada del *Programa NLF* es demasiado bajo, en este caso menos de 3000 muestras (ver Tabla 6 y Figura 31). Sin embargo, a partir de este valor, la gráfica empieza a exhibir un comportamiento algo lineal, cuando el tiempo de procesado empieza a superar al de sobrecarga. Se han obtenido las medidas de tiempo en el Ordenador 1 (Tabla 6 y Figura 31).

En la *Estrategia de* `parfeval` observamos que la función `parfeval` hace que la ejecución tarde aproximadamente una vez y media lo que tardaba el original, produciendo el efecto contrario del deseado (ver Tabla 7). En la Figura 32, puede observarse que las dos curvas presentan un comportamiento exponencial, aunque la resultante de aplicar la función `parfeval` presenta una pendiente más pronunciada. En este caso, las medidas de tiempos han sido obtenidas en el Ordenador 1 (Tabla 7 y Figura 32).

En la *Estrategia de Optimización Aplicada al* Programa CWT, se observa que para pocos canales, el tiempo de ejecución consigue reducirse en torno a un 60% (ver Tabla

8). Puede apreciarse en el salto de 32 canales a 64, cómo la curva de tiempos originales presenta un comportamiento exponencial, mientras que la curva de tiempos del programa optimizado consigue mantener su comportamiento lineal hasta los 64 canales (ver Figura 33). Para finalizar, entre 64 canales y 128, la curva de tiempos optimizados empieza a presentar comportamiento exponencial, la diferencia de tiempo apenas aumenta y se va haciendo cada vez más constante. Para un número de 128 canales de entrada, el tiempo se reducía apenas un 20%. Tras haber implementado los tres cambios descritos con anterioridad, se han obtenido estos tiempos en el Ordenador 2 (Tabla 8 y Figura 33).

Si volvemos a mirar al análisis realizado por la función `profile`, vemos que los tiempos de ejecución se han reducido notablemente. La función que calcula CWT ahora tarda tan solo 18 segundos (Figura 34); asimismo, cuando se salvan los datos del *Programa CWT*, el proceso baja de los 40 segundos de retardo (Figura 35) y la función de filtrado necesita tan solo 1 segundo (Figura 36).

Tabla 6. Tiempos obtenidos con código vectorizado y spmd *para la Estrategia de* spmd *y vectorización.*

Número de muestras	**Tiempo original (s)**	**Tiempo de ejecución tras aplicar la optimización de la *Estrategia de `spmd` y Vectorización* (s)**		
		Vectorizado (s)	**Vectorizado y `spmd` (s)**	**Mejora de rendimiento respecto de vectorizado (%)**
1000	2,87	0,34	17,37	-5008,82
2000	10,68	1,08	18,51	-1613,89
3000	25,49	2,23	18,61	-734,53
4000	34,21	3,78	19,01	-402,91
5000	54,27	5,94	24,83	-318,01
6000	74,11	9,37	31,03	-231,16
7000	105,03	11,20	42,14	-276,25
8000	140,59	14,29	50,94	-256,47
9000	180,49	19,00	67,79	-256,79
10000	238,83	24,72	78,30	-216,75

5. RESULTADOS

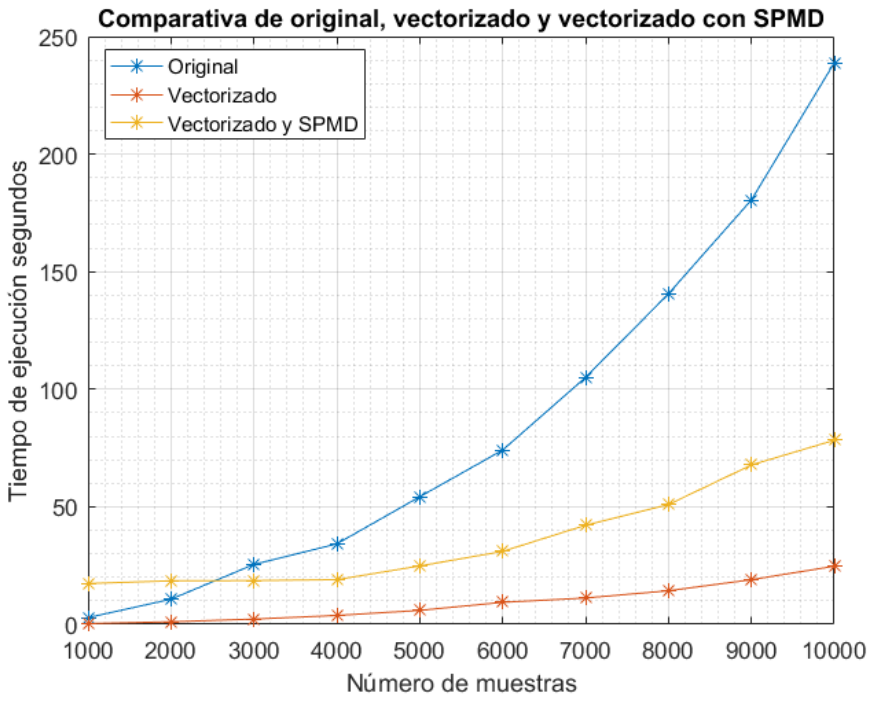

Figura 31. Gráfica con la comparativa entre vectorizado simple y vectorizado con spmd.

Tabla 7. Tiempos obtenidos al aplicar al código la función parfeval *mediante la* Estrategia de *parfeval.*

Número de muestras	**Tiempo de ejecución tras aplicar la optimización de la *Estrategia de `parfeval`* (s)**		
	Original (s)	**`Parfeval` (s)**	**Mejora de rendimiento (%)**
1000	2,87	3,75	-30,54
2000	10,68	12,27	-14,86
3000	25,49	29,56	-15,98
4000	34,21	49,04	-43,35
5000	54,27	76,22	-40,45
6000	74,11	110,10	-48,56
7000	105,03	147,60	-40,53
8000	140,59	225,94	-60,71
9000	180,49	254,99	-41,28
10000	238,83	333,50	-39,64

5. RESULTADOS

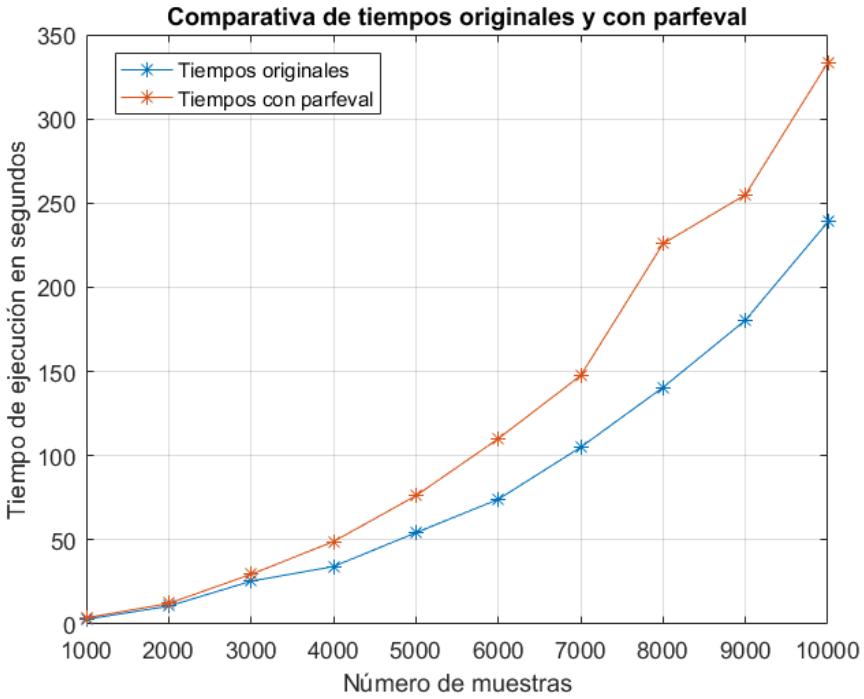

Figura 32. Gráfica comparativa entre versión original y versión con parfeval *mediante la* Estrategia de *parfeval.*

Tabla 8. Tiempos obtenidos tras implementar las mejoras introducidas en el Programa CWT *mediante la* Estrategia de Optimización Aplicada al *Programa CWT* .

Canales	**Tiempo media original (s)**	**Tiempo tras aplicar la optimización de la *Estrategia de Optimización Aplicada al* Programa CWT (s)**	**Mejora de rendimiento (%)**
17	160,35	56,15	64,98
32	230,55	99,09	57,02
64	547,31	191,66	64,98
128	1897,91	1463,09	22,91

5. RESULTADOS

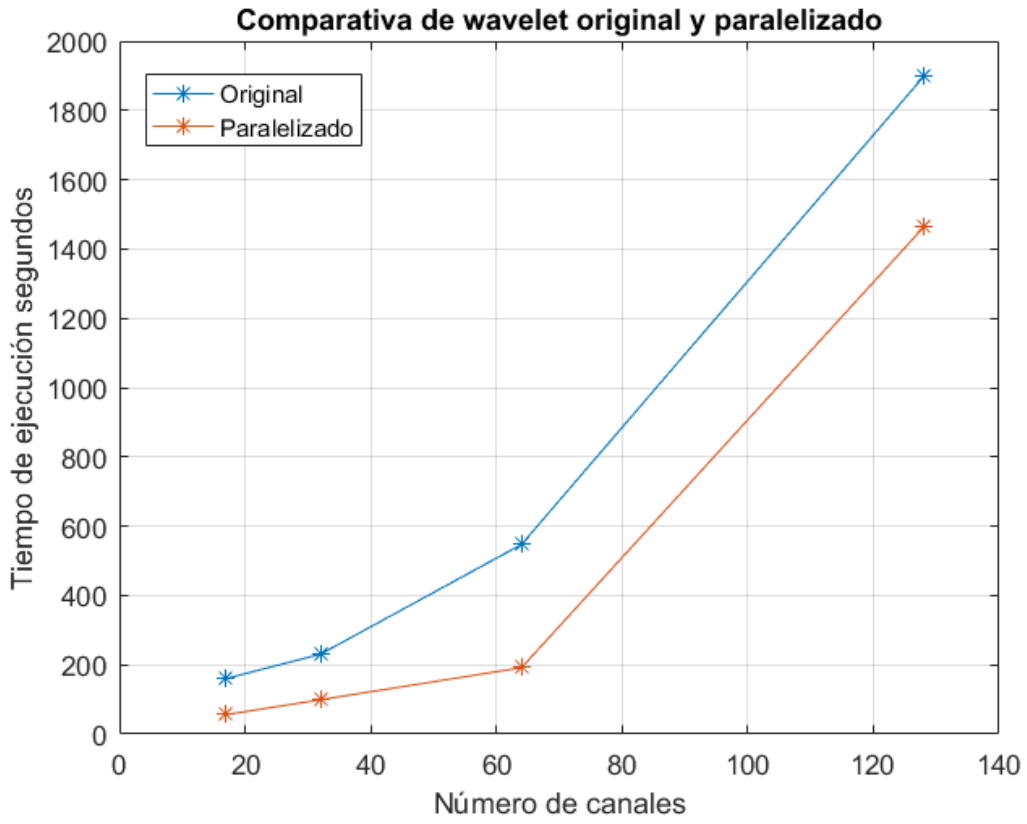

Figura 33. Comparativa de tiempos antes y después de paralelizar el Programa CWT.

```
18.33        1   248      CWTData =  CalcularCWT_NST(cleanEstimulo,pseudo_escala,wav_madre);
```

Figura 34. Profile del Programa CWT *optimizado (I).*

```
38.13       1  276    parfor i=1:N
                  277        psdsliced=Cpsd(:,:,:,i);
                  278        currentfile=sprintf('psd%d.mat',i);
                  279        %save([dir_ppal '\CWT17\' registros(nreg).name(1:5)],'Cpsliced','-v7.3');
                  280        parsave(dir_ppal, currentfile ,psdsliced);
                  281    end
                  282     %save([dir_ppal '\CWT17\' Psd],'Cpsd','-v7.3','-nocompression');
< 0.01      1  283    save([dir_ppal '\CWT17\' registros(nreg).name(1:5)],'-append','Cscales','-nocompression');
< 0.01      1  284    save([dir_ppal '\CWT17\' registros(nreg).name(1:5)],'-append','Cfrequency','-nocompression');
< 0.01      1  285    save([dir_ppal '\CWT17\' registros(nreg).name(1:5)],'-append','Cresolucion_t','-nocompression');
< 0.01      1  286    save([dir_ppal '\CWT17\' registros(nreg).name(1:5)],'-append','Cresolucion_f','-nocompression');
< 0.01      1  287     save([dir_ppal '\CWT17\' registros(nreg).name(1:5)],'-append','Cname','-nocompression');
```

Figura 35. Profile del Programa CWT *optimizado (II).*

```
1.43        1   95    matriz_sujeto = FiltrarDatos_NST(filtro_banda,dir_capado,registros(nreg).name(1:5));
                  96    %delete(gcp('nocreate'));
                  97 %     Se guardan los datos filtrados
```

Figura 36. Profile del Programa CWT *optimizado (III).*

5.3. ANÁLISIS DE RESULTADOS

La *Estrategia de Vectorización* es de las más eficientes, ya que en vez de ir utilizando los elementos uno a uno en un bucle se utilizan todos a la vez, lo cual implica algo de sacrificio en memoria aunque se consigue mayor rapidez de ejecución. Es una solución muy escalable.

Los resultados obtenidos para la *Estrategia de Bucle Paralelo* son también bastante satisfactorios en lo que a tiempo de ejecución se refiere. Hay que destacar que los valores de tiempo para el *Programa NLF* original y vectorizado cambian, al haberse repetido las ejecuciones en una máquina distinta; sin embargo, los valores relativos apenas se ven modificados. Al subir el número de procesos ejecutándose en el servidor, se consigue terminar la tarea en menos tiempo. Las Estrategias de vectorización y de bucle paralelo dan lugar a resultados muy similares, aunque la técnica de la *Estrategia de Vectorización* sigue siendo algo más eficiente.

La *Estrategia de GPU*, que implicaba el uso de una GPU, se ha descartado debido a los grandes retardos que introducen las funciones `gpuArray` y `gather` al

estar contenidas en un bucle de 10 iteraciones. Estas funciones pueden ayudar bastante a la hora de procesar datos en una GPU, pero no cuando se utilizan de forma repetida en el código, ya que introducen mucha sobrecarga de transferencia de datos entre la tarjeta y el cliente, como ocurre en este caso.

En la *Estrategia de GPU y Vectorización* se puede afirmar que procesar este programa en una GPU no es lo más adecuado. Las mismas razones de la *Estrategia de GPU* valdrían para justificar la ineficiencia en este nuevo caso. La vectorización ayuda a compensar este retardo, aunque sigue siendo más efectiva por sí sola.

En la *Estrategia de* `spmd` hay mucha sobrecarga introducida por los métodos `distributed` y `gather`, al estar contenidos en un bucle de 10 iteraciones. Este caso es análogo al de la GPU. Las cláusulas `spmd` trabajan con datos distribuidos, para que los trabajadores puedan comunicarse entre sí, por lo que necesitamos las funciones `distributed` y `gather` para transferir los datos entre el *cluster* y el cliente, aunque si se invocan demasiadas veces, será demasiado coste de ejecución para el programa, que tardará más en terminar.

En la *Estrategia* `spmd` *y Vectorización*, se intentó compensar de alguna manera el retardo causado por las funciones de transferencia de datos entre el cliente y los *workers,* añadiendo la *Estrategia de Vectorización*, contrarrestando, en parte, pero no del todo, dichas tardanzas, por lo que el método de vectorización sigue siendo más efectivo por sí solo por el momento.

La *Estrategia de* `parfeval` tampoco es muy efectiva en este escenario, ya que la sentencia que queremos acelerar, que invoca a la función `sort`, no sobrecarga lo suficiente. Por lo tanto, al aplicarle esta estrategia su tiempo de ejecución aumenta más, a diferencia de otras sentencias, cuya ejecución implique más recursos de computación y, por tanto, se beneficien de la implementación optimizada.

La *Estrategia de Optimización Aplicada* al Programa CWT, conseguía optimizar en gran medida los casos de 16, 32 y 64 canales. Probablemente sea este el límite de escalabilidad de esta solución. Se utilizan bucles paralelos para calcular la CWT y se almacenan los valores de la PSD obtenida dividida en fragmentos de forma paralela, que era la principal causa de sobrecarga del código, en cuya versión original se pretendía guardar todas las variables de una vez. Además, al utilizar compresión por motivos de eficiencia de memoria, todavía contribuía más a ralentizar el programa. Se sigue manteniendo la compresión solo para la PSD, que es el dato de salida más problemático por su volumen. El espacio ocupado por el resto de datos de salida es mucho menor y no es comparable al de la PSD, por lo que no precisan de técnicas de compresión.

En resumen, se han probado varias herramientas para intentar paralelizar nuestros programas, aunque muchas de ellas no han arrojado resultados satisfactorios, bien por la naturaleza de nuestro código o bien por ser demasiado complejas de utilizar. Las dos herramientas más adecuadas para nuestro caso han sido los bucles paralelos y la vectorización, que han resultado ser muy útiles para reducir la carga computacional del *Programa CWT*.

5. RESULTADOS

6. CONCLUSIONES

6.1. CUMPLIMIENTO DE LOS OBJETIVOS

Al principio de este TFG se expusieron aquellos objetivos que se perseguían. El objetivo principal consistía en optimizar dos códigos utilizados por el GIB de la UVA. Se ha cumplido tras haber hecho las pruebas pertinentes y ver que, para un número de canales bajos, el tiempo se reduce de forma considerable, aunque se ha encontrado menor eficacia a partir de un número de canales del orden de cien.

En relación a los objetivos específicos:

- El primer objetivo consistía en la catalogación de herramientas y técnicas de paralelización y sí que se ha logrado, al contar con la suficiente bibliografía y la documentación oficial de MATLAB® disponible en Internet.
- La siguiente meta que nos habíamos propuesto consistía en identificar aquellas partes de los *scripts* que sobrecargaran más que las otras y también se ha cumplido. Se ha empleado la herramienta de MATLAB® `profile`, que ha sido útil para identificar aquellas partes que son más susceptibles de mejorarse.
- El siguiente cometido era probar las diferentes estrategias recogidas en nuestra lista de estrategias. Se han probado distintas opciones de paralelización y optimización a lo largo de las mismas, lo que ha permitido evaluar la complejidad de cada una y la adecuación a los casos de estudio.
- Tras la implementación de varias estrategias y tras la obtención y análisis de los tiempos, se ha decidido que los dos mejores métodos para los escenarios de optimización planteados son la vectorización y los bucles paralelos. Por tanto, junto con `profile`, se han tomado como herramientas para optimizar el *Programa CWT*, cumpliendo de esta manera con la última pauta inicial. Otras alternativas han sido descartadas bien por limitación de hardware, bien por dificultad excesiva en su uso o por inviabilidad.

6.2. PRINCIPALES IDEAS EXTRAÍDAS

Basándonos en los objetivos iniciales de este TFG y su grado de cumplimiento, se obtienen las siguientes conclusiones.

- A pesar de la cantidad de herramientas de las que dispone el *Parallel Computing Toolbox* de MATLAB®, se han acabado utilizando muy pocas en este TFG, ya que la utilidad de cada herramienta dependía de varios factores como su facilidad de uso, los recursos disponibles y, por supuesto, el grado de adecuación a nuestro caso.
- A la hora de medir retardos de ejecución de nuestros programas, han sido dos factores los que ha habido que tener muy en cuenta para obtener resultados

coherentes y precisos. El primero de ellos es que no basta con medir el retardo una vez para un contexto determinado, ya que cualquier factor puede hacer que el resultado sea distinto del esperado. Por lo tanto, se necesitan realizar más medidas para posteriormente promediarlas. El segundo de ellos es que no todas las máquinas procesan a la misma velocidad. Por ello los tiempos absolutos para una misma simulación, pero distintos terminales, pueden ser dispares. Esto implica que todos los datos de cada comparativa han de extraerse de una misma máquina y utilizar tiempos relativos.

- Tras haber paralelizado el segundo programa usado por el GIB de la UVA *(Programa CWT),* y tras la pertinente batería de pruebas, se ha observado que para un número de entre 17 y 64 canales el tiempo de ejecución se reduce en un porcentaje de entre el 55-65%. Sin embargo, cuando se evaluó para el caso de 128 electrodos, la efectividad se reducía de forma considerable a un 22% de mejora de rendimiento. Por tanto, podría considerarse una implementación más óptima a partir de estas cantidades.

6.3. LIMITACIONES Y LÍNEAS FUTURAS

El desarrollo del presente estudio no ha estado exento de limitaciones y restricciones que pueden haber condicionado los resultados presentados:

- En una misma máquina se obtenían medidas de tiempo distintas para una misma simulación en idénticas condiciones, variando entre 5 y 10 segundos, por lo que se deduce que no solo depende de la capacidad de los procesadores en los que se ejecuta el código, sino que hay otros factores que influyen como puede ser su temperatura o el número de procesos ejecutándose en ese momento.
- A lo largo del estudio se ha ido observando la variación del tiempo de ejecución como indicador de si es buena técnica de paralelización. Sin embargo, no se ha tenido en cuenta el cómo afecta a la memoria una vez que se ha modificado el programa, que por lo general ha de aumentar. En futuros estudios se podría estudiar la relación entre las variaciones de tiempo y memoria que permitan alcanzar, no solo un mayor grado de optimización, sino un mejor compromiso también entre estas dos variables.
- No siempre se ha utilizado un mismo equipo para la obtención de muestras, al no contar algunos con el *hardware* necesario para simular todas las estrategias.
    - Un terminal contaba con solo 4 *cores*, por lo que se podían crear hasta 4 *workers*, mientras que otros tenían 6, lo que permitía trabajar con un número máximo 6 *workers*.
    - Algunos ordenadores utilizados no contaban con una GPU suficientemente potente y su memoria se desbordaba fácilmente. Ha habido que hacer uso de otras tarjetas y equipos más potentes para cumplir con nuestro cometido.

  Con el fin de obtener resultados coherentes con todos los datos de una misma estrategia obtenidos de una misma máquina, ha habido que repetir ciertas simulaciones. Lo ideal hubiera sido un mismo equipo durante todo el estudio

con los suficientes componentes, aunque era difícil de prever desde un primer momento.

- Todos los datos de las distintas estrategias se han obtenido como el promedio de 5 repeticiones idénticas para el mismo código y los mismos tamaños de entrada. Se podría haber alcanzado un mayor grado de exactitud si el número de iteraciones hubiera sido mayor, como 10, 20 o incluso 100. No obstante, se ha elegido este valor para no alargar algunas de las ejecuciones más pesadas durante varios días o semanas y por dar un aspecto de uniformidad a todo el estudio.
- Se ha realizado el estudio con longitudes de entrada de hasta 10000 muestras para el *Programa NLF* y números de hasta 128 canales para el *Programa CWT*. Sin embargo, en futuros estudios relacionados con este TFG, podrían trabajarse con valores más elevados para observar las tendencias y verificar las hipótesis de escalabilidad, así como mejorar el diseño de optimización.

6. CONCLUSIONES

7. BIBLIOGRAFÍA

Aftanas, L. I., Lotova, N. V., Koshkarov, V. I., Popov, S. A., & Makhnev, V. P. (1997). Nonlinear forecasting measurements of the human EEG during evoked emotions. *Brain Topography*, *10*(2), 155–162. https://doi.org/10.1023/A:1022208012586

Blinowska, K. J., & Malinowski, M. (1991). Non-linear and linear forecasting of the EEG time series. *Biological Cybernetics*, *66*(2), 159–165. https://doi.org/10.1007/BF00243291

Bronzino, J. D. (2006). *The Biomedical Engineering Handbook*. (J. D. Bronzino, Ed.). Boca Raton, Florida.

Martín González, A. (2015). *Caracterización de la actividad neuronal en pacientes con depresión mediante medidas de acoplamiento*. Trabajo Fin de Grado, Universidad de Valladolid.

Mathworks. (2015). Parallel Computing Toolbox Documentation. Retrieved May 23, 2018, from http://se.mathworks.com/help/distcomp/

Ploskas, N. (2016). *Parallel Computing Toolbox*. (N. Ploskas & N. Samaras, Eds.) (1st ed.). MK. https://doi.org/10.1016/B978-0-12-805132-0.00003-5

Poza, J. (2008). *Análisis tiempo-frecuencia de la actividad magnetoencefalográfica espontánea en la enfermedad de Alzheimer*. Tesis Doctoral, Universidad de Valladolid.

Sharma, G., & Martin, J. (2009). MATLAB : A Language for Parallel Computing. *Int J Parallel Prog*, *37*, 5–7. https://doi.org/10.1007/s10766-008-0082-5

Vajteršic, M., Zinterhof, P., & Trobec, R. (2009). *Overview - Parallel computing: Numerics, applications, and trends*. (P. Zinterhof, Ed.), *Parallel Computing: Numerics, Applications, and Trends*. London. 2-52 https://doi.org/10.1007/978-1-84882-409-6_1

Yunfan, G., Jianxue, X., Wei, R., Sanjue, H., & Fuzhou, W. (1998). Biological Cybernetics. *Biological Cybernetics*, *165*, 159–165.

7. BIBLIOGRAFÍA

Apéndice A. ABREVIATURAS Y ACRÓNIMOS

CWT Transformada *wavelet* continua (*Continous Wavelet Transform*)

EEG Electroencefalograma

GIB Grupo de Ingeniería Biomédica

GPU Unidad de Procesamiento Gráfico (*Graphics Processing Unit*)

MAU Unidad de Acceso a Memoria (*Memory Access Unit*)

MEG Magnetoencefalograma

MJS *Scheduler* de trabajos de MATLAB® (*MATLAB® Job Scheduler*)

PSD Densidad espectral de potencia (*Power Spectral Density*)

TFG Trabajo Fin de Grado

UVA Universidad de Valladolid

Apéndice A. ABREVIATURAS Y ACRÓNIMOS

Apéndice B. PLIEGO DE CONDICIONES

Este anexo contiene las condiciones legales que guiarán a la realización del TFG titulado como 'Estudio del Toolbox "*Parallel Computing*" de MATLAB®. Aplicación a Métodos de Procesado de Señal'.

En lo que sigue se supondrá que el trabajo ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de utilizar el software MATLAB®. Dicha empresa ha debido desarrollar una línea de investigación con el objeto de llevar a cabo el trabajo. Esta línea de investigación, junto con el posterior desarrollo de la aplicación, está amparada por las condiciones particulares del siguiente pliego. Supuesto que la utilización industrial de los métodos recogidos en el presente TFG ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes condiciones:

Condición 1

La modalidad de contratación será el concurso. La adjudicación se hará por tanto a la proposición más favorable, sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho de declararlo desierto.

Condición 2

El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

Condición 3

En la oferta se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

Condición 4

La obra se realizará bajo la dirección técnica de un Graduado en Ingeniería de Telecomunicación o de un Ingeniero Superior de Telecomunicación.

Condición 5

Aparte del Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no está obligado a aceptarla.

Condición 6

El contratista tiene derecho a sacar copias a su costa del software, pliego de condiciones y presupuestos. El Graduado en Ingeniería autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

Condición 7

Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le haya comunicado por escrito el Ingeniero

Director de obras siempre que dicha obra se haya ejecutado a los preceptos de los pliegos de condiciones, de acuerdo a los cuales se harán modificaciones y la valoración de las diversas unidades, sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto no podrán servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

Condición 8

Tanto en las certificaciones de la obra como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

Condición 9

Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata, pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero considere justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

Condición 10

Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o

materiales análogos si los hubiere, y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento se sujetarán siempre al establecido en el punto anterior.

Condición 11

Cuando el contratista, con la autorización del Ingeniero Director de obras, emplee material de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio, ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sino a lo que le correspondería si se hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

Condición 12

Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ella se formen, o en su defecto, por lo que resulte de su medición final.

Condición 13

El contratista queda obligado a abonar al Graduado en Ingeniería autor del proyecto y director de obras, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

Condición 14

Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

Condición 15

La garantía definitiva será del 4% del presupuesto y la provisional del 2%.

Condición 16

La forma de pago será por certificaciones mensuales de la obra ejecutada de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

Condición 17

La fecha de comienzo de las obras será a partir de los quince días naturales del replanteo oficial de las mismas, y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna a la reclamación de la fianza.

Condición 18

Si el contratista, al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

Condición 19

El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras o con el delegado que este designe, para todo lo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

Condición 20

Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista la conservación de la obra ejecutada hasta la recepción de la misma, por lo que el deterioro total o parcial de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o construido por su cuenta.

Condición 21

El contratista deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa por retraso en la ejecución siempre que este no sea debido a causas de fuerza mayor. A la terminación de la obra se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

Condición 22

Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta

Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas establecidas.

Condición 23

Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad "Presupuesto de Ejecución por contrata", y anteriormente llamado "Presupuesto de Ejecución Material" que hoy designa otro concepto. La empresa constructora, que ha desarrollado este proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares.

Condición particular 1

La propiedad intelectual de los procesos descritos y analizados en el presente trabajo pertenece por entero a la empresa constructora representada por el Ingeniero Director del Proyecto.

Condición particular 2

La empresa constructora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos posteriores, para la misma empresa cliente o para otra.

Condición particular 3

Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con la autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

Condición particular 4

En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

Condición particular 5

En todas las reproducciones se indicará su procedencia, explicando el nombre del proyecto, nombre del Ingeniero Director y empresa consultora.

Condición particular 6

Si el proyecto pasa a la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de este, la empresa consultora decidirá aceptar o no la modificación propuesta.

Condición particular 7

Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta al añadirla.

Condición particular 8

Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

Condición particular 9

Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

Condición particular 10

La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

Condición particular 11

La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia de este hecho. En este caso deberá autorizar expresamente los proyectos presentados por otros.

Condición particular 12

El Ingeniero Director del proyecto será responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona asignada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.

Apéndice B. PLIEGO DE CONDICIONES

Los requisitos, clasificados como documentación, hardware y software, que han sido necesarios para la elaboración del presente TFG son los siguientes.

DOCUMENTACIÓN

Servicio de la Biblioteca de la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad de Valladolid.

Mathworks®.

Tesis y Trabajos Fin de Grado de miembros del GIB de la UVA.

HARDWARE

Ordenador portátil con Intel® Core(TM) i7-6700HQ CPU @ 2.60 GHz 2.60 GHz, RAM de 8 GB y 4 *cores.* (Ordenador 1)

Ordenador de sobremesa con Intel® Core(TM) i7-3930K CPU @ 3.20 GHz 3.20 GHz, RAM de 32 GB y 6 *cores.* (Ordenador 2)

Ordenador de sobremesa con Intel® Xeon E5-1620v4 CPU @ 3.50 GHz 3.50 GHz, RAM de 8 GB y 4 *cores.* (Ordenador 3)

Tarjeta GPU NVIDIA® Titan XP

Memoria extraíble 8 GB.

CD-ROM.

Disco extraíble 2 TB.

SOFTWARE

Microsoft® Windows 10.

Microsoft Office 2013. Versión 15.0.5031.1000

MATLAB® Student R2016b. Versión 9.1.0.441655

Adobe® Acrobat Reader®. Versión 2018.011.20038

Google Chrome. Versión 66.0.3359.139

Mendeley. Versión 1.17.13

Apéndice B. PLIEGO DE CONDICIONES

Apéndice C. PRESUPUESTO REQUERIDO

Ejecución material

Ordenador portátil con Intel® Core(TM) i7-6700HQ CPU @ 2.60 GHz 2.60 GHz, RAM de 8 GB y 4 *cores* (Ordenador 1).....800 €

Ordenador de sobremesa con Intel® Core(TM) i7-3930K CPU @ 3.20 GHz 3.20 GHz, RAM de 32 GB y 6 *cores* (Ordenador 2)….....6000 €

Ordenador de sobremesa con Intel® Xeon E5-1620v4 CPU @ 3.50 GHz 3.50 GHz, RAM de 8 GB y 4 *cores* (Ordenador 3)…….....8000 €

Tarjeta GPU NVIDIA® Titan Xp.....1.300 €

Memoria extraíble 8 GB…….....5 €

Disco extraíble 2 TB……..50 €

Impresora…….....40 €

Software……1.045 €

TOTAL EJECUCIÓN MATERIAL……..….17.240 €

Gastos generales

16% sobre la ejecución material……..2.758,40 €

Beneficio industrial

6% sobre la ejecución material……1.034,40 €

Material fungible

Gastos de impresión……..75 €

Gastos de encuadernación…….45 €

CD-ROM…….....10 €

TOTAL MATERIAL FUNGIBLE……..130 €

Honorarios del proyecto

450 horas a 30 € la hora…….....13.500 €

Subtotal del proyecto……...30.870 €

I.V.A. aplicable

21% del subtotal del proyecto…….6.482,70 €

TOTAL DEL PROYECTO……...35.372,70 €

El total del presupuesto asciende a:

TREINTA Y CINCO MIL TRESCIENTOS SETENTA Y DOS EUROS CON SETENTA CÉNTIMOS DE EURO

En Valladolid, junio de 2018 Fdo: Pablo Marina Boillos