UNIVERSIDADE DE SANTIAGO DE

COMPOSTELA

ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

PARALELIZACIÓN Y OPTIMIZACIÓN DE ALGORITMOS DE STEREO MATCHING

Autor:
Aleixo Cambeiro Barreiro

Directores:
Juan Carlos Pichel Campos
Francisco Fernández Rivera

Grao en Enxeñaría Informática

Julio 2015

Traballo de fin de Grao presentado na Escola Técnica Superior de Enxeñaría da Universidade de Santiago de Compostela para a obtención do Grao en Enxeñaría Informática

D. Juan Carlos Pichel Campos, Profesor do Departamento de Electrónica e Computación da Universidade de Santiago de Compostela, e **D. Francisco Fernández Rivera**, Profesor do Departamento de Electrónica e Computación da Universidade de Santiago de Compostela,

INFORMAN:

Que a presente memoria, titulada *Paralelización y optimización de algoritmos de Stereo Matching*, presentada por **D. Aleixo Cambeiro Barreiro** para superar os créditos correspondentes ao Traballo de Fin de Grao da titulación de Grao en Enxeñaría Informática, realizouse baixo nosa dirección no Departamento de Electrónica e Computación da Universidade de Santiago de Compostela.

E para que así conste aos efectos oportunos, expiden o presente informe en Santiago de Compostela, a 8 de Xullo de 2015:

O director, O codirector, O alumno,

Juan Carlos Pichel Campos Francisco Fernández Rivera Aleixo Cambeiro Barreiro

ÍNDICE

1. INTRODUCCIÓN

El tema principal que aborda este trabajo es el Stereo Matching, así como la optimización de algunos de los algoritmos más relevantes de este campo. Por lo tanto, es vital comprender en qué consiste este antes de profundizar en la materia. Además de su naturaleza, también es interesante conocer sus usos principales para entender la importancia de la optimización de rendimiento, objetivo de este estudio.

1.1. Stereo Matching

Se conoce como Stereo Matching o Stereo Vision al proceso de extracción de información sobre la profundidad de los objetos de una escena, basado en la comparación de varias imágenes de esta tomadas desde diferentes puntos. A efectos de ilustración, se adjunta en la Figura 1.1 el siguiente par de imágenes con el correspondiente mapa de profundidad de la escena para la vista de la izquierda:

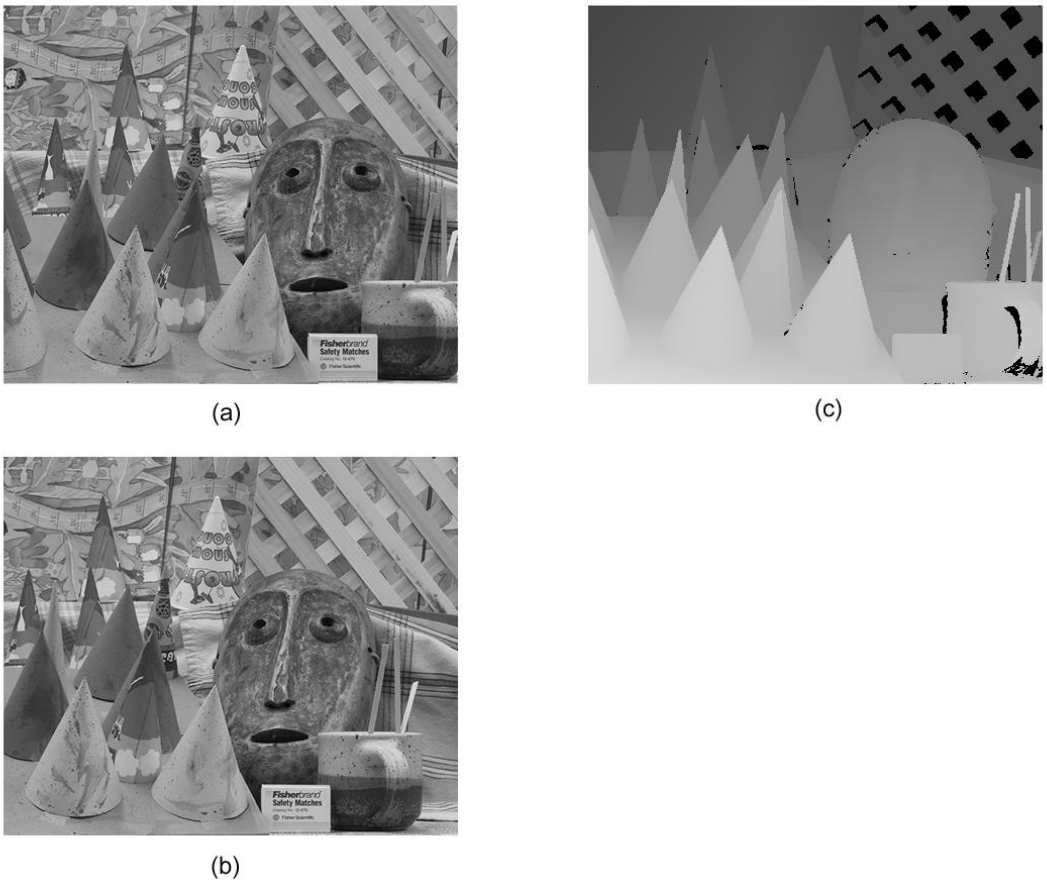


FIGURA 1.1: EJEMPLO DE STEREO MATCHING: (A) IMAGEN BASE (IZQUIERDA), (B) IMAGEN DERECHA, (C) MAPA DE PROFUNDIDAD

Esto se puede conseguir conociendo la posición de las proyecciones de un mismo punto en dos imágenes de la misma escena, así como las posiciones relativas de las cámaras que las han obtenido y sus parámetros intrínsecos (relativos a la lente, deben ser obtenidos mediante calibración experimental de la máquina). Con estos datos podemos deducir la posición del punto original en un espacio tridimensional. Esta nos permite conocer la profundidad del objeto en la escena respecto a la posición de cada cámara.

Existe un factor fundamental que permite simplificar enormemente este proceso: la geometría epipolar [1]. Esta se basa en el concepto de línea base, la cual se define como aquella que conecta los centros de dos cámaras, y consiste en la geometría de la intersección de los planos que tienen esta recta como eje con los planos de proyección correspondientes a las susodichas cámaras. Dentro de ella se definen tres entidades básicas:

- Epipolo: punto de intersección de la línea base con un plano de proyección.
- Plano epipolar: cualquier plano que contenga la línea base.
- Línea epipolar: intersección de un plano epipolar con un plano de proyección.

Con esto, tenemos la garantía de que, dada una proyección x de un punto X de la escena en una de las imágenes, contenidos ambos en un plano epipolar π, la proyección x' sobre la otra imagen también se encontrará, si existe, en el mismo plano. Esto significa que pertenecerá a la línea epipolar correspondiente a π, como podemos ver en la representación gráfica de la Figura 1.2.

En dicha figura podemos ver los dos planos de formación de imágenes, correspondientes a sendas cámaras, y sobre ellos los elementos arriba definidos: para un punto X se muestra su proyección x en el primero de los planos, y una posible proyección del punto, cuya posición tridimensional es una incógnita, sobre la línea epipolar correspondiente en el segundo plano, l'. También se muestra la línea base entre los dos centros, así como los epipolos (e y e') que esta genera en ambas proyecciones.

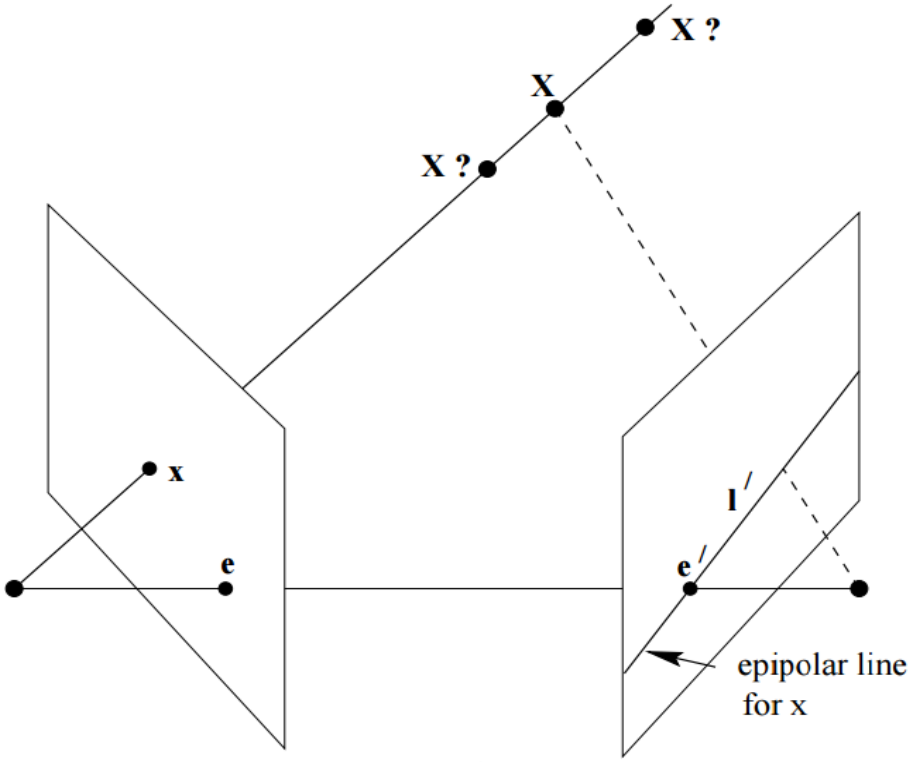


FIGURA 1.2: GEOMETRÍA EPIPOLAR

Los axiomas expuestos implican la posibilidad de ahorrar muchos recursos en la búsqueda de la correspondencia para la proyección de un punto en una de las imágenes, ya que la posición de la otra estará restringida a una recta en lugar de un plano, reduciendo en una dimensión el espacio de búsqueda.

Por otra parte, dadas dos perspectivas de la misma escena, se puede aplicar una transformación proyectiva a una de ellas para simular una rotación de la cámara correspondiente, de modo que la relación entre ambas proyecciones sea solamente de traslación. Esto se puede utilizar para corregir las imágenes que se van a comparar de modo que las líneas epipolares en ambas sean horizontales (con el epipolo en el infinito) y estén a la misma altura, lo que implica otra importante simplificación de los algoritmos de búsqueda de correspondencias: las líneas epipolares se corresponderán con las filas de píxeles.

Dadas estas bases, el desafío que surge es cómo distinguir de entre los posibles candidatos de correspondencia con la proyección de un punto, cuál de ellos es su verdadera pareja, la cual no siempre existe debido a oclusiones o al recortado de las imágenes. Existen numerosas aproximaciones para la resolución de este problema, de entre las cuales se han elegido para abordar en este estudio el método *Pixel-wise*

Matching [2], dos variantes del *Patch-wise Matching* [3] y el *Semi-Global Matching* [2], por su importancia didáctica o su representatividad del estado del arte del campo.

1.2. Usos del Stereo Matching e importancia de su optimización

Entre muchas otras, algunas de las aplicaciones más comunes pueden ser [4]:

- Seguimiento tridimensional: contar personas o monitorizar sus trayectorias, lo que podría tener usos, por ejemplo, en sistemas de seguridad y vigilancia.
- SLAM (*Simultaneous Location and Mapping*): movimiento en un entorno desconocido, utilizando características particulares como marcas para construir un mapa y localizarse en él de forma simultánea.
- Navegación de robots autónomos: detección fiable de obstáculos para el trazado de trayectorias viables.
- Asistencia a la movilidad para los visualmente impedidos: detección e identificación de obstáculos que complemente al uso de un bastón.

En todas ellas se deben obtener soluciones en tiempo real.

En otros ámbitos también se utiliza, por ejemplo, para realizar mapas de contorno o, de forma reseñable, para reconstrucciones tridimensionales de escenarios. En el caso de las aplicaciones en tiempo real, la necesidad de la optimización de estos algoritmos es evidente: para obtener unos tiempos de respuesta válidos es importante realizar el procesamiento de la forma más eficiente posible. Por lo demás, un cálculo más rápido siempre es deseable, y en cualquiera de los casos anteriores, ya sea trabajando con flujos de imágenes o grupos predefinidos de ellas, la cantidad de datos que deben ser tratados es muy elevada, de modo que pequeñas optimizaciones podrían resultar en un gran impacto en el rendimiento.

1.3. Objetivos del proyecto

Se han definido cuatro objetivos principales en este proyecto, que son los que se listan a continuación:

- **OBJ-01**: Revisión de los algoritmos más eficientes y modernos para la resolución del problema del Stereo Matching.
- **OBJ-02**: Adaptación de estos algoritmos a arquitecturas paralelas *multi-*core y many-*core*.
- **OBJ-03**: Optimización del rendimiento de estos algoritmos en las arquitecturas utilizadas.
- **OBJ-04**: Comparación de las versiones optimizadas con los algoritmos base.

1.4. Arquitecturas consideradas

Se han considerado como candidatos para la ejecución de estos algoritmos las siguientes arquitecturas:

- **Intel Xeon E5-2630Lv2** [5]: Procesador multinúcleo para computación de altas prestaciones.
- **Intel Xeon Phi 7120P** [6]: Coprocesador con gran nivel de paralelismo debido a su elevado número de núcleos.
- **Placa *Parallella*** [7]: Microprocesador embebido en una placa dispuesta para la ejecución paralela *multi-core* de bajo consumo.

Los dos sistemas de Intel se encuentran instalados en el mismo servidor el cual consta de 2 CPUs Intel Xeon y un coprocesador Intel Xeon Phi.

Por otra parte, la realización de medidas sobre la placa *Parallella* ha tenido que ser descartada por dificultades técnicas en su puesta en marcha debidas principalmente a la escasa documentación disponible sobre ella.

En el Capítulo 6 se tratarán en profundidad estas arquitecturas, explicando con un mayor nivel de detalle sus características, así como las de la estación de trabajo en la que se encuentran.

2. GESTIÓN DEL PROYECTO

En este capítulo se abordan los aspectos principales de la gestión realizada sobre el proyecto, exponiendo los resultados de los análisis realizados y las decisiones tomadas a lo largo de este proceso.

2.1. Alcance

En este punto se describe lo que se espera como resultado de la consecución de este proyecto.

2.1.1. Descripción del alcance

El producto resultante del desarrollo del proyecto será un software que implemente varios algoritmos de Stereo Matching en sus versiones secuenciales básicas, así como versiones optimizadas de los mismos, permitiendo obtener para pares de imágenes pre-procesadas el mapa de profundidad correspondiente. Las versiones optimizadas consistirán en códigos paralelos adaptados para su ejecución en sistemas many-core. También realizará medidas sobre los resultados que permitan comparar las versiones base de los mencionados algoritmos con aquellas que introduzcan las mejoras.

Además, el producto incluirá un manual de uso del susodicho software y los resultados de un análisis comparativo que explique las mejoras incorporadas.

2.1.2. Criterios de aceptación del producto

En base a los objetivos establecidos para el proyecto, se podrá lograr la aceptación final del producto cuando se cumplan los siguientes criterios básicos:

- El software implementa diversas aproximaciones a la resolución del problema del Stereo Matching, tanto en sus formas simples secuenciales como en versiones optimizadas paralelas
- El software proporciona mejorías en el rendimiento en las versiones adaptadas, basadas en las mencionadas optimizaciones del código
- Se ha realizado un estudio que aporta una cuantificación y análisis de estas mejorías sobre diversas arquitecturas *many-core*

2.1.3. Productos entregables del proyecto

El proyecto cuenta con dos productos principales que se deben entregar tras su finalización, que enumeramos a continuación:

- **Software funcional** con las características anteriormente descritas.
- **Memoria del proyecto** que incluya tanto los aspectos de gestión del proyecto como el estudio de los resultados y un manual de usuario para el **software funcional**.

2.1.4. Exclusiones del proyecto

En este estudio nos hemos centrado en la obtención de profundidades relativas dentro de la escena: en un rango de valores arbitrario, sin conexión con medidas reales, restringiendo el problema al Stereo Matching puro. Para ello hemos trabajado con imágenes ya pre-procesadas, de modo que las líneas epipolares coinciden, como en el ejemplo teórico del primer punto introductorio, con las filas de píxeles y, además, se proporciona un rango máximo de disparidad obtenido de forma experimental, que limita la distancia de búsqueda: solamente se busca a lo largo del fragmento de la línea epipolar definido por este rango.

De este modo, no se trata el pre-procesado de las imágenes utilizadas para las medidas, extraídas de uno de los benchmarks más utilizados en el campo [8]. Tampoco se trata la utilización de los resultados en forma de profundidades relativas.

2.1.5. Restricciones del proyecto

Existe una restricción principal del proyecto a la hora de desarrollar el software y es que este deberá correr sobre las arquitecturas elegidas que serán máquinas cuyo sistema operativo es Linux.

2.1.6. Supuestos del proyecto

Este proyecto se desarrolla bajo el supuesto de que las imágenes extraídas del benchmark para la realización de medidas están correctamente calibradas, pues el susodicho benchmark está respaldado por uno de los centros más prestigiosos de la actualidad en lo referente a Stereo Matching.

2.2. Casos de uso

Los casos de uso describen los posibles escenarios de interacción de los diferentes actores del sistema con el propio software. Capturan el comportamiento deseado del sistema sin especificar cómo implementarlo, por lo que resultan muy útiles para modelar los requisitos funcionales.

En la figura 2.1 se incluye un diagrama que muestra de forma gráfica los casos de uso identificados, así como los actores que en ellos participan:

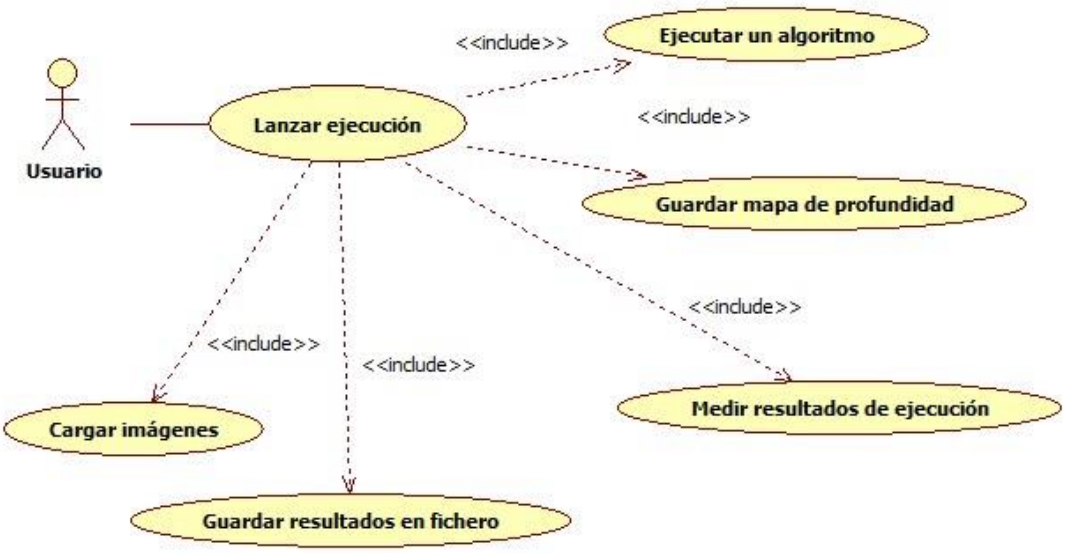


FIGURA 2.1: DIAGRAMA DE CASOS DE USO

A continuación, se incluye una especificación de los casos de uso. Para clasificarlos según su utilidad e importancia se utilizarán las escalas descritas en las siguientes tablas:

Escala de estabilidad	
Baja	El caso de uso se basa en una idea que todavía no se ha desarrollado por completo, por lo que aún no se comprende su contenido en profundidad. Según avanza el proyecto, es probable que cambie.
Media	Los contenidos del caso de uso están claros, aunque su alcance no está bien delimitado. Podría cambiar a lo largo del proyecto.
Alta	El caso de uso tiene un alcance y unos contenidos bien definidos. Es poco probable que cambie durante el desarrollo del proyecto.

Escala de importancia	
Baja	El caso de uso complementa a otros más importantes para acercar ligeramente el producto a la consecución de sus objetivos.
Media	El caso de uso ayuda de por sí a acercar sensiblemente el producto a la consecución de sus objetivos.
Alta	El caso de uso es crucial para que el producto cumpla con sus objetivos.

Las especificaciones de los casos de uso son las siguientes:

ID	**UC-01**

<table>
<tr><td>Nombre</td><td>Lanzar ejecución</td></tr>
<tr><td>Descripción</td><td>El sistema deberá comportarse como se describe en el siguiente caso de uso cuando un usuario trate de ejecutar el programa.</td></tr>
<tr><td>Precondición</td><td>Ninguna.</td></tr>
<tr><td>Secuencia normal</td><td>
<table>
<tr><th>Paso</th><th>Acción</th></tr>
<tr><td>1</td><td>El usuario establece los parámetros de ejecución que utilizará el algoritmo que vaya a lanzar vía línea de comandos.</td></tr>
<tr><td>2</td><td>El sistema utilizará esos parámetros para cargar las imágenes correspondientes.</td></tr>
<tr><td>3</td><td>El sistema utilizará los parámetros para lanzar el algoritmo elegido en la versión requerida.</td></tr>
<tr><td>4</td><td>El sistema realizará las medidas pertinentes sobre la ejecución del algoritmo.</td></tr>
<tr><td>5</td><td>El sistema guardará el mapa de profundidad en el destino elegido.</td></tr>
<tr><td>6</td><td>El sistema guardará los resultados de las mediciones en el destino elegido, si se ha estipulado alguno.</td></tr>
</table>
</td></tr>
<tr><td>Postcondición</td><td>Se ha ejecutado uno de los algoritmos implementados, realizado medidas sobre él, guardado el mapa de profundidad resultante y, si así se ha estipulado, los resultados de las mediciones.</td></tr>
<tr><td>Excepciones</td><td>
<table>
<tr><th>Paso</th><th>Excepción</th></tr>
<tr><td>1</td><td>1-A: Si los parámetros elegidos no son válidos, se cancela la ejecución y se muestra un mensaje de error.</td></tr>
<tr><td>2</td><td>2-A: Si no se pueden cargar las imágenes estipuladas, se cancela la ejecución y se muestra un mensaje de error.</td></tr>
<tr><td>5</td><td>5-A: Si no se puede guardar el mapa de profundidad se cancela la ejecución y se muestra un mensaje de error.</td></tr>
<tr><td>6</td><td>6-A: Si no se pueden guardar los resultados se cancela la ejecución y se muestra un mensaje de error.</td></tr>
</table>
</td></tr>
<tr><td>Estabilidad</td><td>Media</td></tr>
<tr><td>Importancia</td><td>Alta</td></tr>
</table>

<table>
<tr><th>ID</th><th>UC-02</th></tr>
<tr><td>Nombre</td><td>Cargar imágenes</td></tr>
<tr><td>Descripción</td><td>El sistema deberá comportarse como se describe en el siguiente caso de uso cuando se vayan a cargar las imágenes elegidas.</td></tr>
<tr><td>Precondición</td><td>Ninguna.</td></tr>
<tr><td>Secuencia normal</td><td>
<table>
<tr><th>Paso</th><th>Acción</th></tr>
<tr><td>1</td><td>El sistema llama a una librería auxiliar para la carga de imágenes.</td></tr>
<tr><td>2</td><td>El sistema almacena las imágenes cargadas en memoria.</td></tr>
</table>
</td></tr>
<tr><td>Postcondición</td><td>Las imágenes de entrada se encuentran en memoria.</td></tr>
<tr><td>Excepciones</td><td>
<table>
<tr><th>Paso</th><th>Excepción</th></tr>
<tr><td>1</td><td>1-A: Si no se pueden cargar las imágenes de entrada, se cancela la ejecución y se muestra un mensaje de error.</td></tr>
</table>
</td></tr>
<tr><td>Estabilidad</td><td>Alta</td></tr>
<tr><td>Importancia</td><td>Alta</td></tr>
</table>

ID	UC-03
Nombre	Ejecutar un algoritmo
Descripción	El *sistema* deberá comportarse como se describe en el siguiente caso de uso cuando se vaya a ejecutar el algoritmo elegido.
Precondición	Se han establecido los parámetros de forma correcta y cargado las imágenes de entrada.

<table>
<tr><td>Secuencia normal</td><td>
<table>
<tr><th>Paso</th><th>Acción</th></tr>
<tr><td>1</td><td>El sistema lanza la ejecución del algoritmo elegido.</td></tr>
<tr><td>2</td><td>El algoritmo procesa las imágenes de entrada en base a los parámetros establecidos y genera un mapa de profundidad como salida.</td></tr>
<tr><td>3</td><td>El sistema almacena en memoria el mapa de profundidad generado.</td></tr>
</table>
</td></tr>
<tr><td>Postcondición</td><td>El mapa de profundidad resultante del procesamiento de las imágenes de entrada de acuerdo a los parámetros establecidos se encuentra en memoria.</td></tr>
<tr><td>Excepciones</td><td>–</td></tr>
<tr><td>Estabilidad</td><td>Alta</td></tr>
<tr><td>Importancia</td><td>Alta</td></tr>
</table>

<table>
<tr><th>ID</th><th>UC-04</th></tr>
<tr><td>Nombre</td><td>Medir rendimiento de la ejecución</td></tr>
<tr><td>Descripción</td><td>El sistema deberá comportarse como se describe en el siguiente caso de uso cuando se ejecute el algoritmo elegido.</td></tr>
<tr><td>Precondición</td><td>Ninguna.</td></tr>
<tr><td>Secuencia normal</td><td>
<table>
<tr><th>Paso</th><th>Acción</th></tr>
<tr><td>1</td><td>El sistema lanza la ejecución del algoritmo elegido con los parámetros seleccionados.</td></tr>
<tr><td>2</td><td>El sistema mide los resultados de la ejecución.</td></tr>
<tr><td>3</td><td>El sistema almacena en memoria los resultados de la ejecución.</td></tr>
</table>
</td></tr>
<tr><td>Postcondición</td><td>Las medidas realizadas sobre la ejecución del algoritmo se encuentran en memoria.</td></tr>
<tr><td>Excepciones</td><td>–</td></tr>
<tr><td>Estabilidad</td><td>Alta</td></tr>
<tr><td>Importancia</td><td>Alta</td></tr>
</table>

<table>
<tr><th>ID</th><th>UC-05</th></tr>
<tr><td>Nombre</td><td>Guardar mapa de profundidad</td></tr>
<tr><td>Descripción</td><td>El sistema deberá comportarse como se describe en el siguiente caso de uso cuando se vaya a almacenar el mapa de profundidad resultante de la ejecución de un algoritmo.</td></tr>
<tr><td>Precondición</td><td>El sistema dispone del mapa de profundidad en memoria y se ha establecido la ruta donde debe ser almacenado.</td></tr>
<tr><td>Secuencia normal</td><td><table><tr><th>Paso</th><th>Acción</th></tr><tr><td>1</td><td>El sistema llama a una librería auxiliar para almacenar el mapa de profundidad como una imagen en el sistema de archivos.</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>Se ha almacenado el mapa de profundidad resultante del procesamiento de un par de imágenes mediante uno de los algoritmos como una imagen en el sistema de archivos.</td></tr>
<tr><td>Excepciones</td><td><table><tr><th>Paso</th><th>Excepción</th></tr><tr><td>1</td><td>1-A: Si no se puede realizar una escritura en la ruta especificada, se cancela la ejecución y se muestra un mensaje de error.</td></tr></table></td></tr>
<tr><td>Estabilidad</td><td>Alta</td></tr>
<tr><td>Importancia</td><td>Alta</td></tr>
</table>

<table>
<tr><th>ID</th><th>UC-06</th></tr>
<tr><td>Nombre</td><td>Guardar resultados en fichero</td></tr>
<tr><td>Descripción</td><td>El sistema deberá comportarse como se describe en el siguiente caso de uso cuando se vayan a volcar a un fichero las mediciones realizadas sobre la ejecución de un algoritmo.</td></tr>
<tr><td>Precondición</td><td>El sistema dispone de las mediciones en memoria y se ha seleccionado la opción de guardarlas en un archivo, estableciendo la ruta del fichero donde han de ser volcadas.</td></tr>
<tr><td>Secuencia normal</td><td><table><tr><th>Paso</th><th>Acción</th></tr><tr><td>1</td><td>El sistema imprime en formato legible los resultados de las mediciones en un archivo de texto en la ruta especificada.</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>Las medidas realizadas sobre la ejecución del algoritmo se encuentran en formato legible en el archivo de texto correspondiente a la ruta especificada.</td></tr>
</table>

Excepciones	<table><tr><th>Paso</th><th>Excepción</th></tr><tr><td>**1**</td><td>1-A: Si no se puede realizar una escritura en la ruta especificada, se cancela la ejecución y se muestra un mensaje de error.</td></tr></table>
Estabilidad	Media
Importancia	Media

2.3. Requisitos

En base a los objetivos fijados y a las reuniones realizadas con los tutores del proyecto, se ha identificado una serie de requisitos que habrá de cumplir el software que se quiere desarrollar para que cumpla con los criterios de aceptación establecidos. Estos se dividen en dos grandes categorías: funcionales y no funcionales.

Para su clasificación, se le ha asignado un identificador a cada uno, consistente en el prefijo **RF** para los funcionales y **RNF** para los no funcionales, seguidos del número correspondiente. También se aporta para cada uno una breve descripción y el nivel de importancia para el proyecto, que puede ser **Necesario**, si se considera imprescindible para la consecución del proyecto, o **Deseable**, si no es imprescindible pero ayudaría a mejorar la calidad del producto.

2.3.1. Requisitos funcionales

Los requisitos funcionales representan aquellas funcionalidades o tareas que se espera que el software desarrollado vaya a cumplir. Se listan a continuación los que se han identificado para este proyecto:

ID	**RF-01**
Descripción	Implementación del algoritmo de Stereo Matching conocido como *Pixel-wise Matching*, así como de su versión optimizada.
Importancia	Necesario

ID	**RF-02**
Descripción	Implementación del algoritmo de Stereo Matching conocido como *Patch-wise Matching* en su versión basada en SAD (Sum of Absolute Differences), así como de su versión optimizada.
Importancia	Necesario

ID	**RF-04**

Descripción	Implementación del algoritmo de Stereo Matching conocido como *Patch-wise Matching* en su versión basada en ASD (Adaptive Support Weight), así como de su versión optimizada.
Importancia	Necesario

ID	**RF-05**
Descripción	Implementación del algoritmo de Stereo Matching conocido como *Semi-Global Matching*, así como de su versión optimizada.
Importancia	Necesario

ID	**RF-06**
Descripción	Lectura y carga de las imágenes utilizadas como entrada para la ejecución de los algoritmos de Stereo Matching.
Importancia	Necesario

ID	**RF-07**
Descripción	Escritura en el sistema de archivos de las imágenes resultantes de la aplicación de los distintos algoritmos.
Importancia	Necesario

ID	**RF-08**
Descripción	Obtención de medidas de rendimiento de la ejecución de cada uno de los algoritmos.
Importancia	Necesario

ID	**RF-09**
Descripción	Posibilidad de escritura en formato legible de los resultados de la ejecución de los algoritmos en un fichero de texto.
Importancia	Deseable

ID	**RF-10**
Descripción	Posibilidad de selección de los parámetros de ejecución de los algoritmos.
Importancia	Necesario

ID	**RF-11**
Descripción	Posibilidad de paso de parámetros de ejecución por línea de comandos para poder establecerlos sin necesidad de re-compilación.
Importancia	Deseable

2.3.2. Requisitos no funcionales

Los requisitos no funcionales representan características requeridas del sistema, proceso de desarrollo, servicio prestado o cualquier otro aspecto del desarrollo que no implique específicamente la construcción de una funcionalidad. Se listan a continuación los que se han identificado para este proyecto:

ID	**RNF-01**
Descripción	El software deberá correr de forma correcta sobre las arquitecturas elegidas, que serán sistemas basados en Linux.
Importancia	Necesario

ID	**RNF-02**
Descripción	Las versiones mejoradas de los diferentes algoritmos deberán ofrecer incrementos en el rendimiento respecto a las implementaciones base.
Importancia	Deseable

ID	**RNF-03**
Descripción	El software incluirá un front-end que cubra las funcionalidades básicas del software para lanzarlo desde una interfaz gráfica.
Importancia	Deseable

2.4. Metodología

Antes de transformar estos requisitos en bloques de actividades y asignarles dependencias y flujo temporal, es importante definir la metodología con la que se ha de abordar el proyecto. Para ello, consideraremos las características de este último y analizaremos qué modelo de ciclo de vida se adapta mejor a la presente situación.

El proyecto tiene un enfoque individual, de modo que no existe la necesidad de ponerse de acuerdo con un equipo o explicar los puntos de vista de cada integrante, por lo que no será necesario profundizar demasiado en el diseño o los procesos de ingeniería del software: con que el alumno tenga claro lo que va a desarrollar y cómo, es suficiente.

Por otra parte, a pesar de que no se espera que los requisitos sean cambiantes, dado que el proyecto es en esencia un estudio de investigación y, como tal, carece de cliente que pueda cambiar de opinión frecuentemente; el plazo límite de entrega es rígido y, por lo tanto, un factor limitante considerable. El alcance podría verse limitado por un desarrollo menos veloz de lo previsto, lo que obligaría a adaptar la planificación a los cambios. Estas modificaciones deberían poder realizarse en el menor tiempo

posible ya que, de otro modo, el tiempo invertido en redistribuir la susodicha planificación podría obligar a reducir el alcance de nuevo.

El anterior riesgo de modificación del alcance para adaptarse al plazo podría verse acentuado por el hecho de que se tratará con un campo de investigación en el que el alumno carece de experiencia, utilizando algunas tecnologías con las que no está familiarizado en profundidad, como puede ser OpenMP. También hay que tener en cuenta que se probarán diferentes enfoques de optimización y que, si no se obtienen los resultados esperados, podría ser necesaria la inclusión de algún otro. Por todo lo anterior, no se espera que las estimaciones temporales iniciales sean extremadamente precisas.

Respecto al tamaño, en lo referente al código desarrollado este proyecto podría considerarse como bastante reducido, lo que refuerza lo que ya se había dicho con anterioridad: se espera que no sea necesaria una documentación intensiva y que un estilo de programación ágil, sin mucha carga en los procesos de ingeniería de software, se vea favorecido.

Dadas las características anteriores, se puede concluir que el ciclo de vida que mejor se adaptará al presente proyecto será aquel que permita un desarrollo ágil, ligero en los apartados de ingeniería de software, diseño y documentación, y que permita una planificación flexible y adaptativa en función de la velocidad de avance. Un método iterativo permitiría, además, introducirse en los aspectos desconocidos rápidamente para calcular el tiempo que lleva en realidad su desarrollo y tener lo antes posible una estimación fiable del trabajo restante. Todo esto lleva a pensar que la metodología idónea en este caso es la conocida como Programación Extrema, cuyas características se exponen en el siguiente punto.

2.4.1. Programación Extrema

La Programación Extrema [9] es una metodología ágil que se centra en el desarrollo del código: la filosofía que encierra es la de desarrollo y prueba continuos, adaptando de forma dinámica la planificación cuando los requisitos cambian o la velocidad de avance no es la esperada, utilizando el código como la propia documentación cuando esto sea posible, y reduciendo la carga de los procesos de ingeniería de software.

Este modelo de ciclo de vida software está pensado para el trabajo en equipos pequeños o medianos, y se basa en el seguimiento de buenas prácticas, que se pueden resumir en los siguientes puntos:

- Revisión y mejora continuas del código
- Simplicidad del diseño
- Iteraciones cortas

- Entregas continuas para testeo del avance

El código, por lo tanto, se desarrollará en pequeñas iteraciones que nos permitan verificar y validar los módulos desarrollados antes de incluir otros nuevos, y reducir o aumentar el alcance en función de la velocidad de avance relativa a la planificación, que será ligera y fácilmente adaptable.

2.5. Planificación temporal

En este apartado se aborda la identificación de los bloques de trabajo principales que se han de llevar a cabo para que el proyecto tenga éxito, así como las previsiones para su distribución a lo largo del tiempo. Esto se conseguirá por medio de un esquema del tipo EDT (Estructura de Descomposición del Trabajo) para el primer fin, y un diagrama de Gantt para el segundo.

2.5.1. Estructura de Descomposición del Trabajo

Un esquema de tipo EDT consiste en una representación jerárquica de los principales bloques de actividades necesarias para la correcta compleción del proyecto que se hayan identificado, desglosándolas por niveles hasta llegar a un nivel de granularidad manejable para una planificación temporal.

En la figura 2.2 se adjunta como una imagen la Estructura de Descomposición del Trabajo generada para este proyecto:

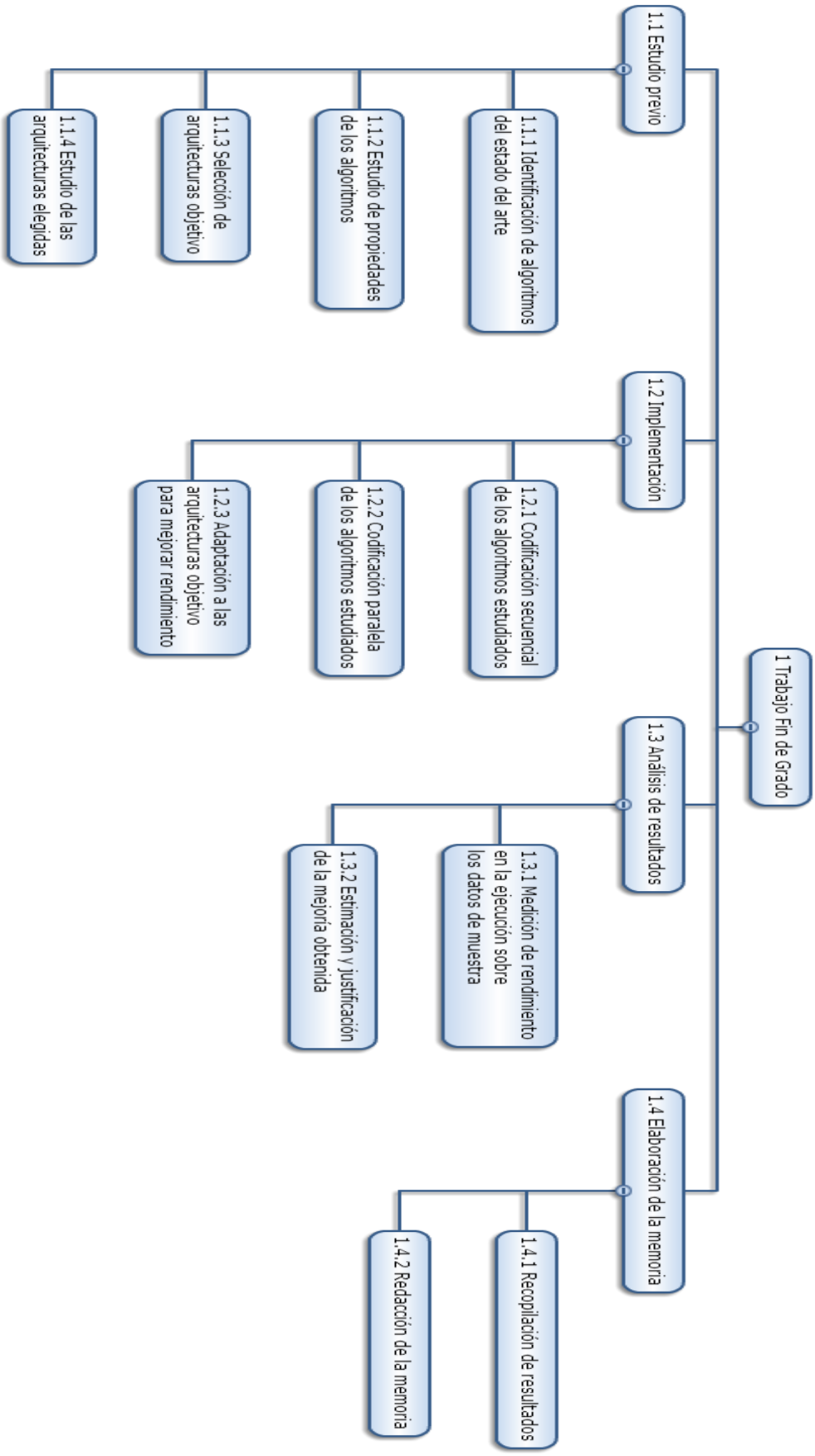


FIGURA 2.2: ESTRUCTURA DE DESCOMPOSICIÓN DEL TRABAJO

A continuación explicamos en qué consiste cada una de las actividades del nivel jerárquico más bajo (con mayor nivel de granularidad) del esquema anterior. El desglose de los costes por cada paquete de trabajo se realiza en el apartado de análisis de costes.

- **1.1.1 Identificación de algoritmos del estado del arte**: Este paquete de trabajo engloba la revisión de la literatura relevante en el campo del Stereo Matching, así como de los benchmarks más importantes para identificar posibles candidatos para su implementación.

- **1.1.2 Estudio de propiedades de algoritmos**: Este paquete de trabajo engloba un análisis sobre cuáles de los algoritmos revisados son más adecuados para su implementación en base a un estudio de sus propiedades.

- **1.1.3 Selección de arquitecturas objetivo**: Este paquete de trabajo engloba la identificación de posibles candidatos para las arquitecturas sobre las que se ejecutará el software desarrollado, así como su estudio para analizar la viabilidad de su uso.

- **1.1.4 Estudio de las arquitecturas elegidas**: Este paquete de trabajo engloba un estudio de las propiedades de las arquitecturas elegidas para la ejecución del software, de modo que se puedan hacer adaptaciones coherentes de los algoritmos implementados.

- **1.2.1 Codificación secuencial de los algoritmos estudiados**: Este paquete de trabajo engloba la implementación básica directa de los algoritmos de Stereo Matching seleccionados para este proyecto.

- **1.2.2 Codificación paralela de los algoritmos estudiados**: Este paquete de trabajo engloba la implementación de diferentes estrategias de paralelización de las versiones secuenciales de los algoritmos, así como la introducción de posibles mejoras de rendimiento.

- **1.2.3 Adaptación a las arquitecturas objetivo para mejorar el rendimiento**: Este paquete de trabajo engloba la implementación de soluciones paralelas específicamente orientadas a alguna de las arquitecturas bajo estudio.

- **1.3.1 Medición de rendimiento en la ejecución sobre los datos de muestra**: Este paquete de trabajo engloba el estudio de las mediciones que se deberían realizar sobre los algoritmos considerados, así como su implementación y ejecución.

- **1.3.2 Estimación y justificación de la mejoría obtenida**: Este paquete de trabajo engloba el análisis de las diferentes mediciones realizadas para estimar en qué medida las soluciones paralelas mejoran el rendimiento de las versiones base en cada arquitectura. También se contempla en este punto la identificación de los motivos que justifican el comportamiento de los resultados obtenidos.

- **1.4.1 Recopilación de resultados**: Este paquete de trabajo engloba la recopilación de los resultados relevantes del estudio realizado y su representación para introducirlos en la memoria del proyecto.

- **1.4.2 Redacción de la memoria**: Este paquete de trabajo engloba la escritura de la memoria del proyecto y de su presentación.

2.5.2. Diagrama de Gantt

Un diagrama de Gantt ofrece una representación temporal para la planificación de las actividades incluidas en los paquetes de trabajo identificados en la Estructura de Descomposición del Trabajo, hasta el nivel de granularidad que se considere necesario para realizar una estimación fiable. En él, además, se muestra la asignación de recursos a cada tarea.

En la figura 2.3 podemos ver un diagrama de Gantt con la planificación realizada para este proyecto:

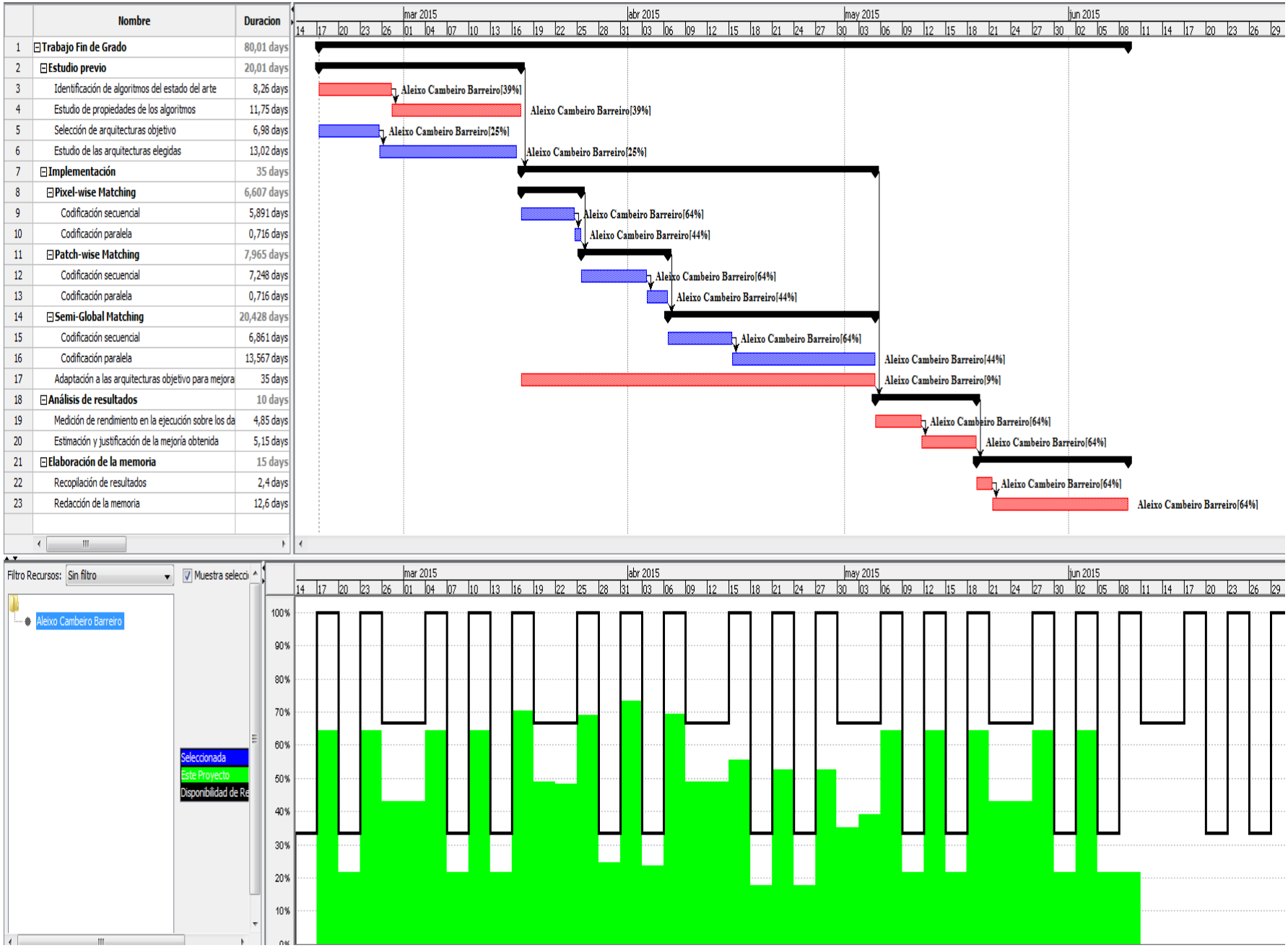


FIGURA 2.3: DIAGRAMA DE GANTT

En el diagrama anterior podemos ver que se ha asignado como único recurso humano a todas las tareas al alumno encargado del proyecto, y que en ningún momento tiene sobreasignaciones, respetando el reparto por semanas estipulado en el anteproyecto.

2.6. Gestión de riesgos

Este apartado está dirigido a la identificación de los riesgos que puedan afectar al proyecto, así como a su categorización y planificación. Para la categorización se utilizarán dos escalas diferentes: la de **impacto** y la de **probabilidad**, con tres valores posibles en ambos casos, que serían Muy alto, Alto y Medio.

ESCALA DE IMPACTO	
Descripción	Repercusión que la concreción del riesgo en una contingencia tendría sobre las posibilidades de completar el proyecto de forma correcta.
Muy alto	El impacto podría tener como consecuencia directa la cancelación del proyecto.
Alto	El impacto podría mermar la calidad del resultado final o causar desórdenes moderados en la planificación.
Medio	El impacto podría alterar ligeramente la planificación o el alcance, sin afectar a la consecución de los objetivos del proyecto.
Bajo	El impacto supondrá retrasos de un orden de magnitud suficientemente reducido como para que no afecten a la planificación.

ESCALA DE PROBABILIDAD	
Descripción	Estimación de la probabilidad que tiene el riesgo de concretarse en una contingencia.
Muy alta	Probabilidad estimada igual o superior al 75%.
Alta	Probabilidad estimada entre el 50% y el 75%.
Media	Probabilidad estimada entre el 25% y el 50%.
Baja	Probabilidad estimada igual o inferior al 25%.

2.6.1. Identificación y categorización

Tras un cuidadoso análisis, se llega a la conclusión de que los riesgos que podrían afectar a este proyecto son los que se listan a continuación:

ID	RS-01
Descripción	Una planificación demasiado optimista hace que no sea posible cumplir el plazo de entrega del proyecto.
Probabilidad	Alta
Impacto	Muy alto
Indicador	En dos revisiones del avance del proyecto se han producido retrasos respecto a la planificación.
Plan de prevención	Reconfigurar la planificación para un mejor aprovechamiento del tiempo de trabajo. Si es necesario, reducir el alcance para hacer el proyecto asequible.

ID	RS-02
Descripción	Una codificación con fallos obliga a una depuración prolongada que retrasa la planificación.
Probabilidad	Alta
Impacto	Alto
Indicador	Se ha dedicado a la depuración de un algoritmo más del 50% del tiempo que se ha tardado en desarrollarlo.
Plan de prevención	Desarrollo de pseudocódigo para los algoritmos antes de su implementación, y revisión grupal con los tutores del proyecto.
Plan de contingencia	Codificar el algoritmo desde 0 o reducir el alcance del proyecto.

ID	RS-03
Descripción	Los planes del proyecto se abandonan por la presión, llevando al caos y a un desarrollo ineficiente.
Probabilidad	Media
Impacto	Muy alto
Indicador	Hace más de dos semanas de la última revisión del avance del proyecto.
Plan de prevención	Ser meticuloso con el cumplimiento de la planificación, realizando revisiones frecuentes y estableciendo un sistema de parámetros de rendimiento que indique acciones correctivas que deben tomarse.
Plan de contingencia	Reducir el alcance del proyecto, volver a planificar y retomar una metodología de desarrollo respetuosa con las previsiones.

ID	RS-04
Descripción	Las partes del proyecto que no se han especificado claramente consumen más tiempo del esperado
Probabilidad	Media
Impacto	Alto
Indicador	Se han producido retrasos respecto a la planificación de dos actividades cuya insuficiente granularidad ha impedido discernir con claridad su duración real.
Plan de	Evitar nombres vagos de tareas en la planificación temporal, y pensar

prevención	detenidamente si cada una se puede resumir mentalmente en una serie de pasos básicos directamente realizables.
Plan de contingencia	Volver a planificar, revisando todas las actividades especificadas para confirmar que la granularidad de descripción es suficiente, y reducir el alcance si es necesario.

ID	**RS-05**
Descripción	La ejecución del software en alguna de las arquitecturas elegidas resulta inviable.
Probabilidad	Alta
Impacto	Alto
Indicador	No se puede ejecutar el software desarrollado en una de las arquitecturas que se habían elegido para este estudio.
Plan de prevención	Revisar las especificaciones de las arquitecturas escogidas y ejecutar las diferentes versiones del software en ellas a medida que se van desarrollando.
Plan de contingencia	Descartar la arquitectura como candidata y elegir otra o reducir el alcance del proyecto si es necesario.

ID	**RS-06**
Descripción	El uso de equipos compartidos para la realización de mediciones de rendimiento, como puede ser el servidor Xeon Phi, resulta en una distorsión de las susodichas medidas.
Probabilidad	Alta
Impacto	Muy alto
Indicador	Las medidas varían en más de un 10% entre distintas ejecuciones en la misma plataforma.
Plan de prevención	Seguir un protocolo de comunicación con los otros usuarios de estos servicios, reservando las máquinas necesarias antes de utilizarlas.

ID	**RS-07**
Descripción	No se obtiene mejoría en el rendimiento mediante las optimizaciones de los algoritmos seleccionados.
Probabilidad	Media
Impacto	Alto
Indicador	Se han realizado medidas sobre las versiones secuencial y paralela de uno de los algoritmos y no se ha obtenido una mejoría significativa (al menos un 10% del tiempo de ejecución).
Plan de prevención	Elegir cuidadosamente los algoritmos que se tomarán como objeto de estudio, teniendo en cuenta que sus propiedades permitan una paralelización eficaz.
Plan de contingencia	Descartar el algoritmo elegido como candidato y elegir otro o reducir el alcance del proyecto.

ID	RS-08
Descripción	La inexperiencia en las tecnologías utilizadas, como OpenMP, así como en el estudio de los algoritmos de Stereo Matching, ralentiza el proceso de desarrollo.
Probabilidad	Media
Impacto	Alto
Indicador	Se ha dedicado más tiempo del estipulado en la planificación al aprendizaje de una de las tecnologías utilizadas en el proyecto.
Plan de prevención	Reducir el uso de tecnologías desconocidas a aquellas que sean necesarias y buscar las alternativas mejor documentadas. Realizar, asimismo, una planificación pesimista del tiempo que hará falta para familiarizarse con ellas.
Plan de contingencia	Tratar de utilizar una alternativa más sencilla para el propósito correspondiente, o reducir el alcance del proyecto si es pertinente y necesario.

2.7. Gestión de la configuración

En este apartado se explica la metodología seguida durante el proyecto en lo que a gestión de la configuración respecta. Esto incluye la identificación de los elementos de configuración y la selección de las herramientas informáticas utilizadas para mantener sus versiones.

Cabe añadir que la función del proceso de gestión de la configuración en este proyecto es meramente la de mantener un control de versiones, ya que no se realizará trabajo de forma concurrente por parte de varias personas: el alumno será el único que modifique los elementos de configuración. Esto se ha tenido en cuenta a la hora de realizar la planificación de la gestión de configuración, adaptándola a los fines que realmente tendrá, aligerando el peso de este proceso durante el desarrollo del proyecto.

2.7.1. Identificación de los elementos de configuración

Los elementos de configuración identificados son los siguientes:

- Código fuente
- Memoria del proyecto
- Planificación del proyecto
- Diseño del software construido

Todos estos elementos, excepto el código fuente, por motivos de simplificación del desarrollo, se regirán por la siguiente nomenclatura:

<Tipo de documento>_<DDMMAA>_<Descripción>_v<Versión>

Los tipos de documentos son los que se recogen en la tabla siguiente:

Tipo de documento	Descripción	Código
Memoria del proyecto	Documento base con la información principal del proyecto desarrollado.	**MDP**
Planificación del proyecto	Documento con la planificación temporal de las actividades del proyecto.	**PTP**
Diseño del software codificado	Documento que incluye diferentes diagramas utilizados para la implementación del software objeto de este proyecto.	**DSC**

2.7.2. Herramientas de gestión de configuración

En la selección del software de soporte a la gestión de configuración para este proyecto, se ha hecho una distinción fundamental entre los elementos de configuración en dos categorías: el código fuente de la aplicación y la documentación.

Para controlar el versionado de los archivos del código fuente, se ha considerado que la herramienta más cómoda es un **repositorio de tipo GIT**; y se ha optado por alojarlo en la web **BitBucket** [10] para simplificar su compartición con los tutores del proyecto. Esto facilita en gran medida tareas como recuperar configuraciones anteriores del código fuente, analizar las diferencias existentes entre ellas o mezclar diferentes versiones de la configuración de forma interactiva.

Por otra parte, para mantener las diferentes versiones de la documentación se ha utilizado **Google Drive** [11], que permite recuperar las últimas versiones guardadas de cada archivo, compartirlas de forma sencilla y mantener una cómoda estructura en carpetas integrada con el sistema de archivos mediante su **cliente de escritorio**. Además, es un sistema gratuito que ofrece una cantidad de espacio de almacenamiento elevada en su versión básica. Se ha prescindido de la utilización de GIT para esta finalidad por varios motivos:

- Es más simple modificar los archivos directamente en el sistema de ficheros, sin perder la posibilidad de recuperar versiones anteriores.

- Las conexiones entre ficheros no son tan importantes como en el código fuente, que es probable que necesite que todos los archivos estén en la versión correspondiente de forma simultánea para poder compilarlos.

- Un repositorio GIT no ayuda con la diferenciación entre versiones de archivos binarios, considerando los ficheros con formato de Word como tales.

2.8. Análisis de costes

En este apartado se analiza el coste económico proveniente de la realización de este proyecto. Se ha planteado como un estudio hipotético en el que el alumno es un trabajador asalariado para obtener una cifra aproximada más realista.

Para este análisis, se han identificado las siguientes fuentes de gastos potenciales:

- **Estación de trabajo**: Para el desarrollo de este proyecto se han utilizado como estaciones de trabajo un ordenador portátil, propiedad del alumno, que no ha supuesto ningún coste adicional por lo tanto; y un equipo de sobremesa adquirido por el CITIUS, por un precio de 1.100€.

- **Servidor de ejecución**: Parte de las pruebas han sido realizadas sobre un servidor Xeon y su coprocesador Xeon Phi, propiedad del CITIUS. Estos ya habían sido adquiridos con otros fines, y su utilización para este proyecto no supone ningún coste adicional. La amortización del equipo supone una cantidad poco significativa y no se incluye en el estudio.

- **Placa *Parallella***: Se ha trabajado con esta placa a lo largo del proyecto, y consiste en un microprocesador multinúcleo portátil. Este ha sido entregado de forma gratuita al CITIUS para su promoción por parte de la empresa vendedora, de modo que no ha supuesto coste alguno.

- **Software utilizado**: Todo el software utilizado para el desarrollo de este proyecto es gratuito, excepto la *suite* de ofimática Microsoft Office 2010 en su versión Home, que estaba instalada en el ordenador portátil del alumno y en su momento costó 119€. No obstante, la amortización de este software a través de su uso para todo tipo de proyectos ha comenzado hace cuatro años, lo que hace que la cantidad que se debería amortizar en este sea despreciable, con lo que no se computará en la suma de costes.

- **Mano de obra**: Como ya se ha mencionado, para aportar realismo a estos datos se ha considerado que el alumno es un trabajador pagado. Para esta estimación, se ha establecido como su rol el de Analista Programador. Tras una investigación en diferentes portales de búsqueda de trabajo, como por ejemplo InfoJobs [12] o Job and Talent [13], se ha establecido que esto supondría un sueldo por hora de aproximadamente 16,51€. El desglose de los costes relativos al uso de esta mano de obra en las diferentes actividades planificadas puede verse en la siguiente tabla extraída del informe realizado de forma automática por el software de planificación:

ID	Actividad	Coste
0	**Trabajo de Fin de Grado**	6.809,55€
1	**Estudio previo**	1.702,59€
1.1	Identificación de algoritmos del estado del arte	429,26€
1.2	Estudio de propiedades de los algoritmos	610,87€
1.3	Selección de arquitecturas objetivo	231,14€
1.4	Estudio de las arquitecturas objetivo	431,32€
2	**Implementación**	2.979,64€
2.1	**Pixel-wise Matching**	542,77€
2.1.1	Codificación secuencial	501,50€
2.1.2	Codificación paralela	41,27€
2.2	**Patch-wise Matching**	658,33€
2.2.1	Codificación secuencial	617,06€
2.2.2	Codificación paralela	41,27€
2.3	**Semi-Global Matching**	1.365,79€
2.3.1	Codificación secuencial	584,04€
2.3.2	Codificación paralela	781,75€
2.4	Adaptación a las arquitecturas objetivo para mejorar el rendimiento	412,75€
3	**Análisis de resultados**	850,27€
3.1	Medición de rendimiento en la ejecución sobre los datos de muestra	412,75€
3.2	Estimación y justificación de la mejoría obtenida	437,52€
4	**Elaboración de la memoria**	1.277,05€
4.1	Recopilación de resultados	203,90€
4.2	Redacción de la memoria	1.073,15€

Esto nos deja con el siguiente desglose de costes para los paquetes de trabajo definidos en la sección del esquema EDT:

- **1.1.1 Identificación de algoritmos del estado del arte**: 429,26€
- **1.1.2 Estudio de propiedades de algoritmos**: 610,87€
- **1.1.3 Selección de arquitecturas objetivo**: 231,14€
- **1.1.4 Estudio de las arquitecturas elegidas**: 431,32€
- **1.2.1 Codificación secuencial de los algoritmos estudiados**: 1.702,59€
- **1.2.2 Codificación paralela de los algoritmos estudiados**: 864.29€
- **1.2.3 Adaptación a las arquitecturas objetivo para mejorar el rendimiento**: 412,75€
- **1.3.1 Medición de rendimiento en la ejecución sobre los datos de muestra**: 412,75€
- **1.3.2 Estimación y justificación de la mejoría obtenida**: 437,52€
- **1.4.1 Recopilación de resultados**: 203,90€
- **1.4.2 Redacción de la memoria**: 1.073,15€

Además de los costes recogidos en los puntos anteriores, hay que tener en cuenta otros gastos indirectos como puede ser el consumo eléctrico, tanto de los servidores como de las estaciones de trabajo utilizadas. Siguiendo las políticas de la USC, se ha decidido calcular estos como un 20% del total [14].

En base a todo lo anterior, se puede deducir la siguiente tabla de costes del proyecto por fuentes potenciales:

Fuente	Coste estimado
Estación de trabajo	1.100€
Servidor de ejecución	0€
Placa *Parallella*	0€
Software utilizado	0€
Mano de obra	6.809,55€
Costes indirectos	1.581,91€
TOTAL	9491,46€

3. HERRAMIENTAS USADAS

En este apartado se analizan las herramientas que se han utilizado durante el desarrollo del proyecto, explicando la funcionalidad de cada una y los motivos de su elección. Separaremos, por lo tanto, las susodichas herramientas en distintas categorías para su tratamiento.

3.1. Herramientas de desarrollo software

Dado que las arquitecturas objetivo para la ejecución del software desarrollado corren sistemas basados en Linux, se ha decidido utilizar este sistema operativo para las labores de construcción del mencionado programa. Se ha optado por la distribución Ubuntu, que es la que utilizan todas las susodichas arquitecturas, para minimizar problemas de compatibilidad. La popularidad de esta distribución, además, garantiza que el producto podrá ser exportado a muchos otros sistemas de forma directa. Este sistema ya se encuentra instalado en la estación de trabajo del CITIUS por defecto, y había sido instalado con anterioridad por el alumno en su ordenador portátil personal.

El sistema operativo utilizado ha condicionado la selección de las herramientas de desarrollo software. Estas han sido las que se listan a continuación:

- **Gedit**: Se ha utilizado este editor de texto para la escritura del código fuente, así como para la visualización de los ficheros de texto con resultados generados por este. Viene instalado de serie con la distribución de Ubuntu y es un editor muy flexible y completo, con todas las funcionalidades necesarias para el desarrollo de software a pequeña escala. Se había considerado como alternativa el IDE NetBeans, pero ha sido descartado porque la versión del repositorio de Ubuntu carece de módulo para desarrollo en C/C++, y la versión descargable desde la web no es muy estable y se ha podido comprobar que tiene fallos ocasionales que hacen muy incómodo utilizarlo. Por otra parte, el software desarrollado es de tamaño reducido, con lo cual no se ha considerado necesaria la utilización de una herramienta IDE para la gestión del desarrollo.

- **G++/Comando *make*/ICC**: En lo referente a la compilación del código generado, se ha optado por estas herramientas. El comando *make* de los sistemas Linux simplifica en gran medida la sistematización de una compilación reiterada con diversos parámetros que no varían, como las librerías utilizadas o el nombre del archivo de output; que pueden agruparse en un *makefile*. Por otra parte, G++ es el compilador más

popular de C++ en sistemas Linux: viene instalado de serie en la distribución Ubuntu, está ampliamente testado, resulta sencillo de utilizar y es muy completo, cubriendo todas las necesidades previstas en el desarrollo de este proyecto. No obstante, para la compilación del código en el procesador Xeon y en el coprocesador Xeon Phi se ha utilizado la herramienta de Intel, ICC, que debería generar un código optimizado para estas arquitecturas y proporciona especialmente muchas facilidades para el desarrollo orientado al Xeon Phi: permite la compilación cruzada desde el Xeon hacia la arquitectura del Xeon Phi para el posterior envío del ejecutable (se comportan como máquinas separadas) y ejecución nativa en el coprocesador; así como la compilación directa con secciones *offload*, cuyo procesamiento recaerá sobre el coprocesador, dejando para la máquina principal el resto de cálculos.

- **Cliente por consola de SSH/SCP**: El alumno no dispone de acceso local directo al servidor que contiene las arquitecturas que se consideran en el trabajo, de modo que el trabajo en esta máquina se realiza mediante acceso remoto utilizando el cliente por consola de SSH que viene instalado de serie en la distribución Ubuntu. Por otra parte, el acceso a Internet desde este servidor está filtrado, y no se permite la conexión al repositorio GIT remoto en BitBucket, por lo que la transferencia del código fuente para compilar se realiza mediante el cliente por consola de SCP que también viene pre-instalado en la citada distribución. Además, como ya se ha dicho, el procesador y el coprocesador funcionan como máquinas diferentes, de modo que para trabajar de forma nativa con el Xeon Phi se realiza un acceso remoto desde el Xeon mediante SSH y para el envío del ejecutable compilado en el procesador principal se utiliza SCP.

- **Red VPN del CITIUS y cliente VPN de Ubuntu**: El acceso remoto a la estación del servidor solamente se puede realizar desde la red interna del CITIUS. Es por ese motivo que para el trabajo desde fuera del CITIUS el alumno ha utilizado el servicio de red VPN del CITIUS, al que se accede a través del cliente VPN integrado en la distribución de Ubuntu utilizada (14.04).

- **Librería OpenMP** [15]: Se ha decidido utilizar el framework OpenMP para el desarrollo de las versiones paralelas de los algoritmos. Es un estándar ampliamente testado y su eficiencia ha sido demostrada. La documentación disponible es muy extensa, y tiene un soporte muy

importante por parte de la comunidad. Su utilización es realmente sencilla ya que, por ejemplo, con un cambio mínimo como puede ser una línea de código con una directiva, se puede paralelizar de forma eficiente un bucle con un reparto equitativo de las iteraciones entre varios hilos, mucho más breve y conciso que alternativas como puede ser la librería POSIX Threads, sin perder por ello opciones de personalización. También se encarga automáticamente de la gestión de muchos aspectos de la paralelización, liberando al programador de esta tarea, que puede muchas veces ser realizada de forma óptima o muy eficiente por la propia máquina.

- **Librería de tratado de imágenes**: Se ha utilizado una librería externa para gestionar la interacción con imágenes: proporciona métodos sencillos para cargarlas en memoria, operar con ellas como matrices de píxeles y guardarlas como ficheros en el sistema de archivos. Se dispone del código fuente, que es compilado como parte del propio programa. Es muy sencilla en su manejo: solamente se necesitan un par de funciones con sintaxis muy clara y el acceso a las matrices de píxeles se ha simplificado mediante el uso de macros. No obstante, cubre todas las necesidades del software desarrollado, reconociendo el formato de imágenes con el que se trabaja (PGM, en escala de grises).

- **GIT**: Como ya se ha explicado en el apartado de gestión de la configuración, para el control de versiones del código fuente se ha utilizado la herramienta GIT, con un repositorio online en la página BitBucket. Para el acceso a este, se han considerado diversos clientes gráficos disponibles en los repositorios de Ubuntu. No obstante, todas estas herramientas han resultado tener una integración bastante pobre con repositorios remotos, por lo que se ha decidido utilizar directamente el cliente por consola, disponible también en el repositorio de Ubuntu. Este es, además, mucho más flexible y completo que las versiones gráficas, y permite un control mayor de las acciones por parte del usuario. A pesar de esto, la carencia de interfaz gráfica resulta inconveniente en algunos puntos, como la revisión del historial de *commits* realizados o la comparación entre diferentes versiones de un mismo archivo o diferentes *commits*. Para suplir estas faltas, se ha utilizado el software Giggle, aunque su uso se ha restringido a front-end de visualización, delegando, como ya se ha dicho, la interacción con el repositorio remoto en la aplicación de consola.

- **Visualizador de imágenes de GNOME**: Para la visualización de los mapas de profundidad resultantes del procesamiento de pares de imágenes de entrada, así como para la de estas últimas, se ha utilizado la herramienta de visualización de imágenes de GNOME, que viene instalada por defecto en la distribución Ubuntu utilizada.

- **NetBeans IDE**: A pesar de que la mayoría del código se ha desarrollado utilizando el editor de texto Gedit y compilación manual por consola por los motivos que ya se han explicado, se ha utilizado esta herramienta para el desarrollo de un front-end en Java Swing para la aplicación. Esto se debe a que las ya mencionadas carencias de la versión para Linux de este entorno de desarrollo integrado no afectan a los módulos de producción en Java. Por lo tanto, se ha considerado que el desarrollo sería mucho más rápido utilizando esta herramienta por las funcionalidades que incorpora, como el auto-completado o la generación automática de código pero, especialmente, por la paleta que permite el desarrollo de interfaces gráficas basadas en Java Swing de forma muy sencilla, directa y visual, con la posibilidad de utilización mediante un sistema de *drag & drop*.

3.2. Herramientas para el desarrollo de la documentación

El desarrollo de la documentación de este proyecto se ha llevado a cabo de forma íntegra en el ordenador portátil personal del alumno. La elección del sistema operativo utilizado con esta finalidad se ha basado en la disponibilidad de herramientas compatibles, así como en la disponibilidad del propio sistema operativo. En el ya mencionado ordenador, se encontraba instalado con anterioridad para otros fines Windows 7 Home Premium, en su versión de 64 bits, el cual ha sido seleccionado como base para el desarrollo de la documentación.

Las herramientas utilizadas para el desarrollo de la documentación de este proyecto son las que se listan a continuación:

- **WBS Tool**: Esta herramienta web se ha utilizado para la generación del esquema de tipo Estructura de Descomposición del Trabajo (WBS por sus siglas en inglés), requerido en la parte de gestión del proyecto. Se ha elegido por la sencillez y agilidad de uso que permite, así como por el hecho de ser una herramienta gratuita que cumple con todas las funciones necesarias: crear una representación gráfica de la descomposición del proyecto en bloques de trabajo, que se pueda

exportar como imagen o como archivo XML, que se podría importar en el futuro desde la misma herramienta para realizar los cambios que sean necesarios, o desde una herramienta de planificación como base para la construcción de un diagrama de Gantt.

- **ProjectLibre**: Se ha utilizado esta herramienta de planificación para añadir flujo temporal y secuencialidad a los bloques de trabajo identificados en el esquema EDT. El criterio principal para su selección ha sido el coste de adquisición, descartando herramientas como el Microsoft Project, tal vez más completas, por su elevado precio, ya que se ha estimado que ProjectLibre cubre con creces todas las necesidades que surgen en la gestión de este proyecto. Por otra parte, esta herramienta también se ha impuesto en la comparativa con otra herramienta gratuita llamada OpenProj, que se puede considerar una versión previa de esta y ofrece funcionalidades más limitadas, con una elevada cantidad de errores que dificultan la realización de una correcta planificación.

- **Microsoft Office 2010**: Se ha utilizado esta suite de ofimática para la redacción de la memoria y la construcción de las gráficas que en ella aparecen, más concretamente las herramientas Word y Excel. Se ha elegido por ser mucho más completa que otras herramientas gratuitas como LibreOffice u OpenOffice, alcanzando niveles más altos de personalización y permitiendo un desarrollo más ágil. No se ha tenido en cuenta el impacto económico, pues ya se disponía de una instalación de este paquete software con anterioridad en el ordenador portátil personal del alumno, de modo que su utilización no ha supuesto ningún coste adicional para el proyecto.

- **StarUML**: Se ha utilizado esta herramienta de diseño para la representación gráfica en forma de esquema de los casos de uso. Es un software gratuito orientado al diseño de software que permite una creación simple de este tipo de diagramas, así como su exportación en forma de archivos de imagen.

- **LucidChart**: Se ha utilizado esta herramienta web para la creación de algunos de los diagramas introducidos en la memoria. Ofrece los componentes gráficos básicos necesarios y tiene una versión gratuita que cubre las funcionalidades requeridas para la elaboración de esta memoria sin necesidad de instalación. Permite guardar los resultados en la nube, en comunicación con una cuenta de usuario de Google a través

de Drive, y exportarlos a diversos formatos para su almacenamiento local. Además de todo esto, su uso es sencillo, mediante una interfaz de estilo *drag & drop* que permite una construcción ágil.

4. ARQUITECTURA DEL SOFTWARE

En este capítulo se discute la arquitectura en la que se ha basado la construcción de este software, identificando los componentes que lo constituyen, así como realizando una descripción de cada uno.

Los citados componentes son: un front-end en Java Swing, la consola de comandos del sistema operativo, un back-end en C++, la estructura para realizar mediciones de rendimiento, el código con los algoritmos de Stereo Matching, una librería externa para el tratamiento de imágenes y el propio sistema de archivos. Su disposición en la construcción del software puede verse en el diagrama de la Figura 4.1:

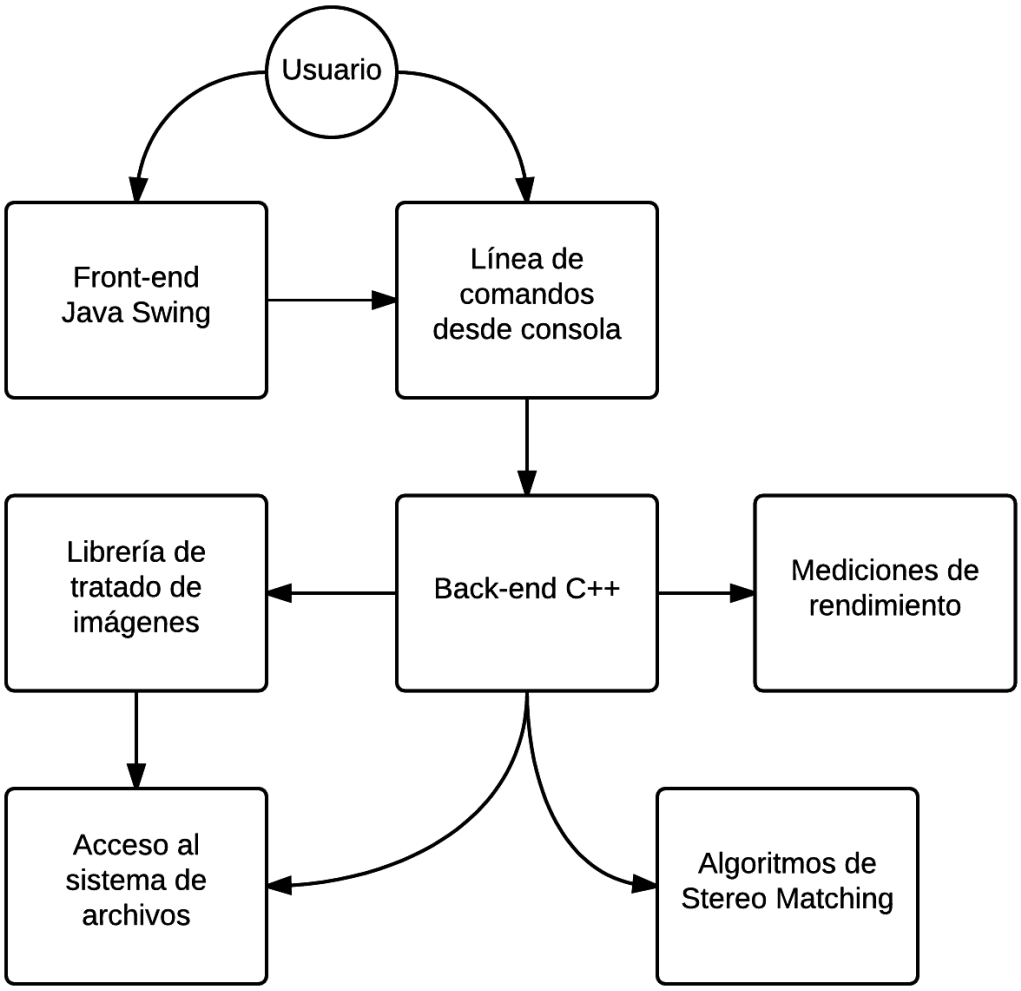


FIGURA 4.1: ARQUITECTURA DEL SOFTWARE DESARROLLADO

4.1. Front-end Java Swing

Se ha decidido que, a pesar de ser suficiente para ejecuciones esporádicas, el método de introducción de parámetros por línea de comandos directamente desde la consola no es práctico para un uso intensivo o por parte de usuarios poco familiarizados con él o con la propia consola, aunque se hayan generado *scripts* para lanzarlo con determinados parámetros de forma automatizada. Por este motivo, se ha optado por la construcción de un front-end para una mayor comodidad de uso y una mayor agilidad en la ejecución sistemática del programa con diferentes opciones. Se ha elegido Java Swing para su construcción por las facilidades que este framework aporta, permitiendo una velocidad de creación de interfaces muy elevada, sobre todo en combinación con el IDE NetBeans y la paleta de componentes que ofrece. Por otra parte, no se ha tenido en cuenta el factor de rendimiento al pensar en la tecnología utilizada para desarrollarlo, ya que solamente es un front-end: lo único que hace es lanzar el back-end, que está escrito en C++ y cuyo rendimiento se ha tratado de optimizar, con determinados parámetros.

En las Figuras 4.2, 4.3, 4.4 y 4.5 podemos ver varias capturas de pantalla que muestran las características de esta interfaz:

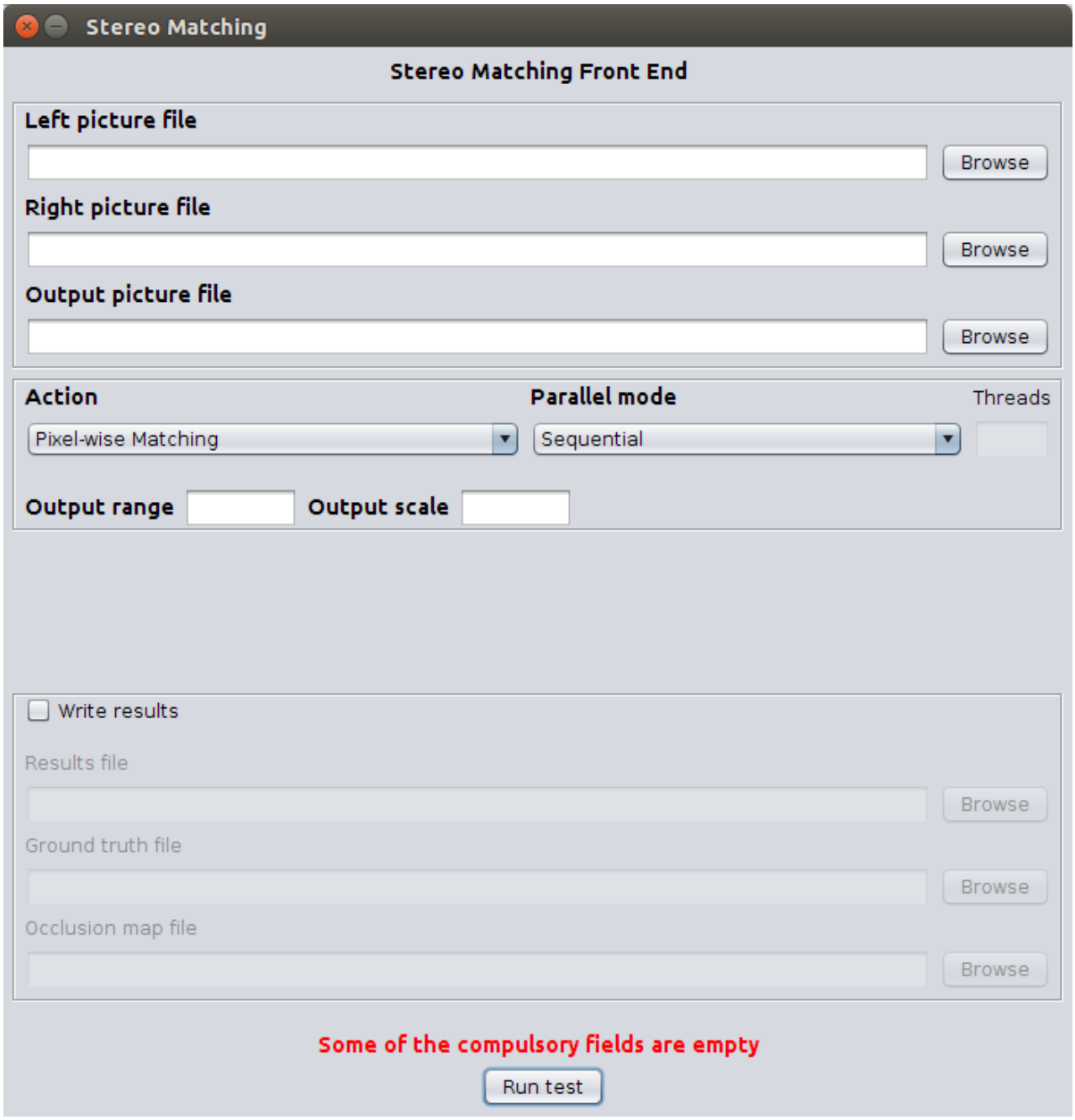


FIGURA 4.2: FORMULARIO DEL FRONT-END VACÍO

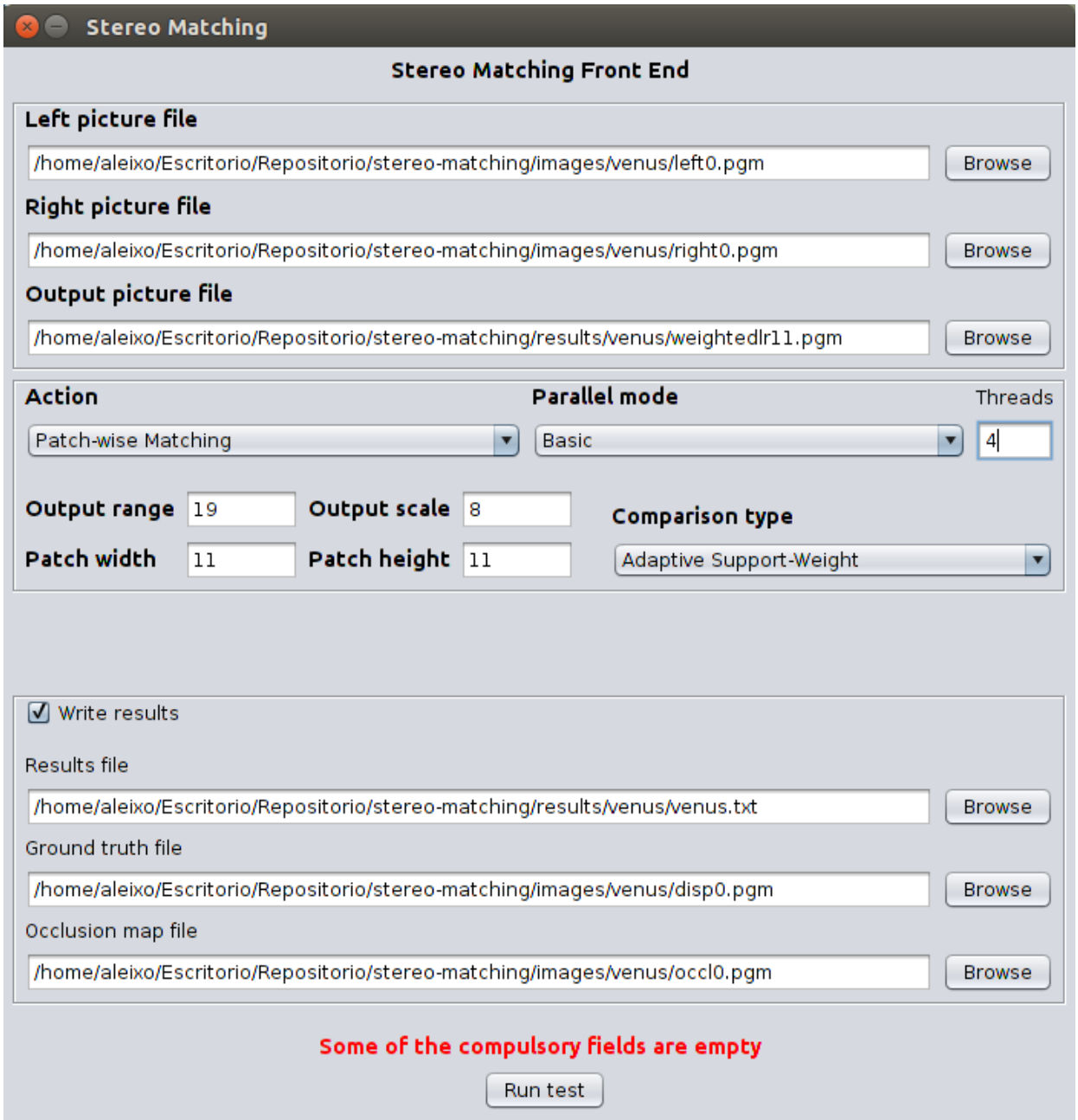


FIGURA 4.3: FORMULARIO DEL FRONT-END CUBIERTO

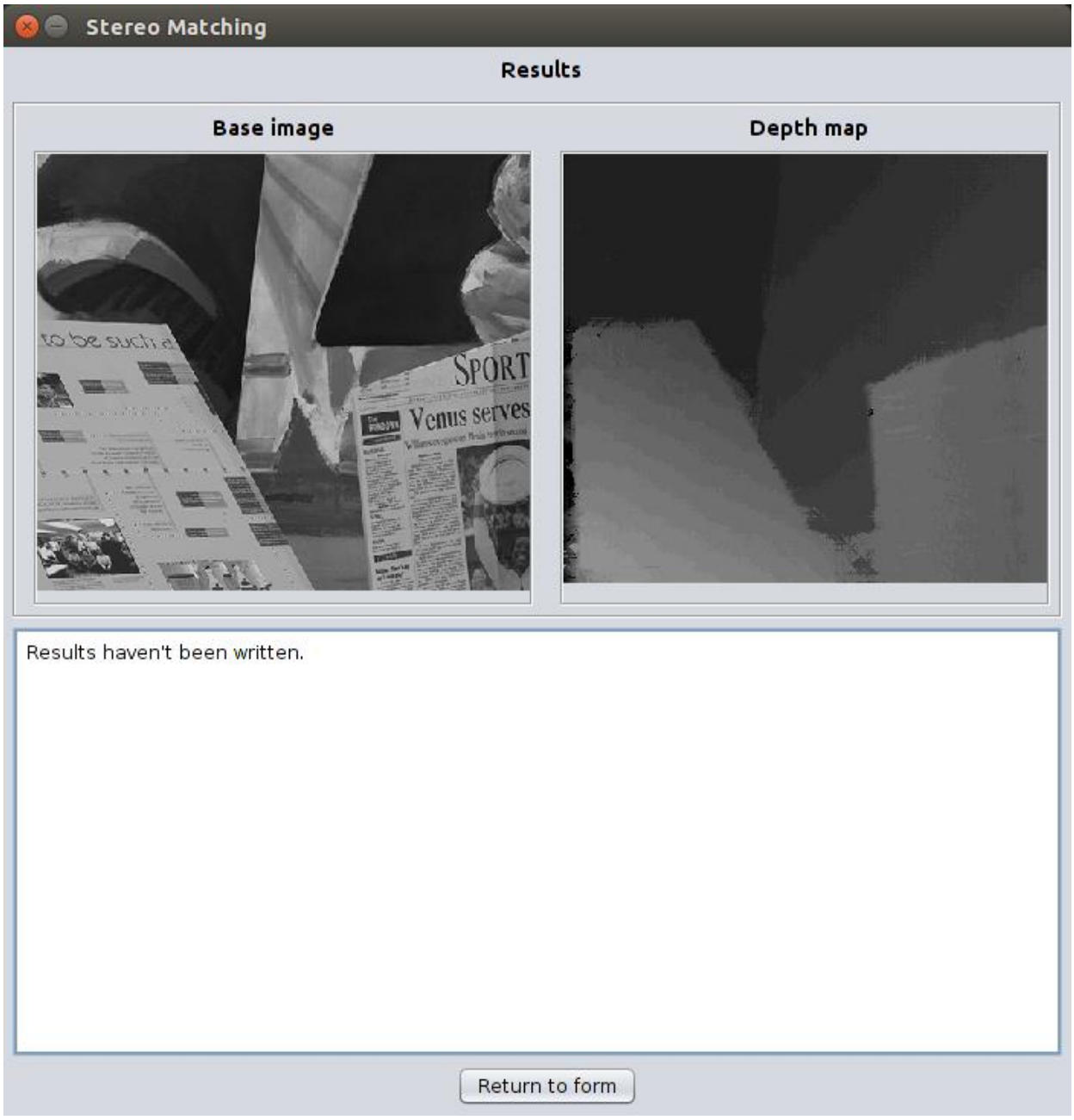


FIGURA 4.4: PANTALLA DE RESULTADOS DEL FRONT-END

FIGURA 4.5: PANTALLA DE RESULTADOS DEL FRONT-END CON DATOS DE RENDIMIENTO

Como se puede ver, se ha optado por un diseño minimalista, mediante un formulario con el mínimo número de campos necesarios para funcionar, sin incluir todas las opciones que ofrece la interacción directa por línea de comandos. También se puede observar que el número de campos visibles varía en función de los parámetros seleccionados.

En lo referente a la pantalla de resultados, como se puede observar, muestra la imagen base del par sobre el que se ha realizado la computación junto al mapa de profundidad generado para ella. Ambas imágenes se escalan para que encajen en los correspondientes paneles, respetando las proporciones entre ancho y alto originales.

Por otra parte, se incluye un área de texto en la que, en caso de haber seleccionado la opción de almacenar los resultados de la ejecución, estos se muestran en un formato de texto legible.

4.2. Línea de comandos desde consola

Como ya se ha mencionado, esta es la opción que permite una máxima flexibilidad de uso, a costa de una menor agilidad y comodidad. Incluye todas las posibles opciones de personalización diseñadas para lanzar los algoritmos, como se puede observar en la pantalla de ayuda de la Figura 4.6 que se ofrece al hacer una introducción de parámetros que no se adapta a la sintaxis esperada:

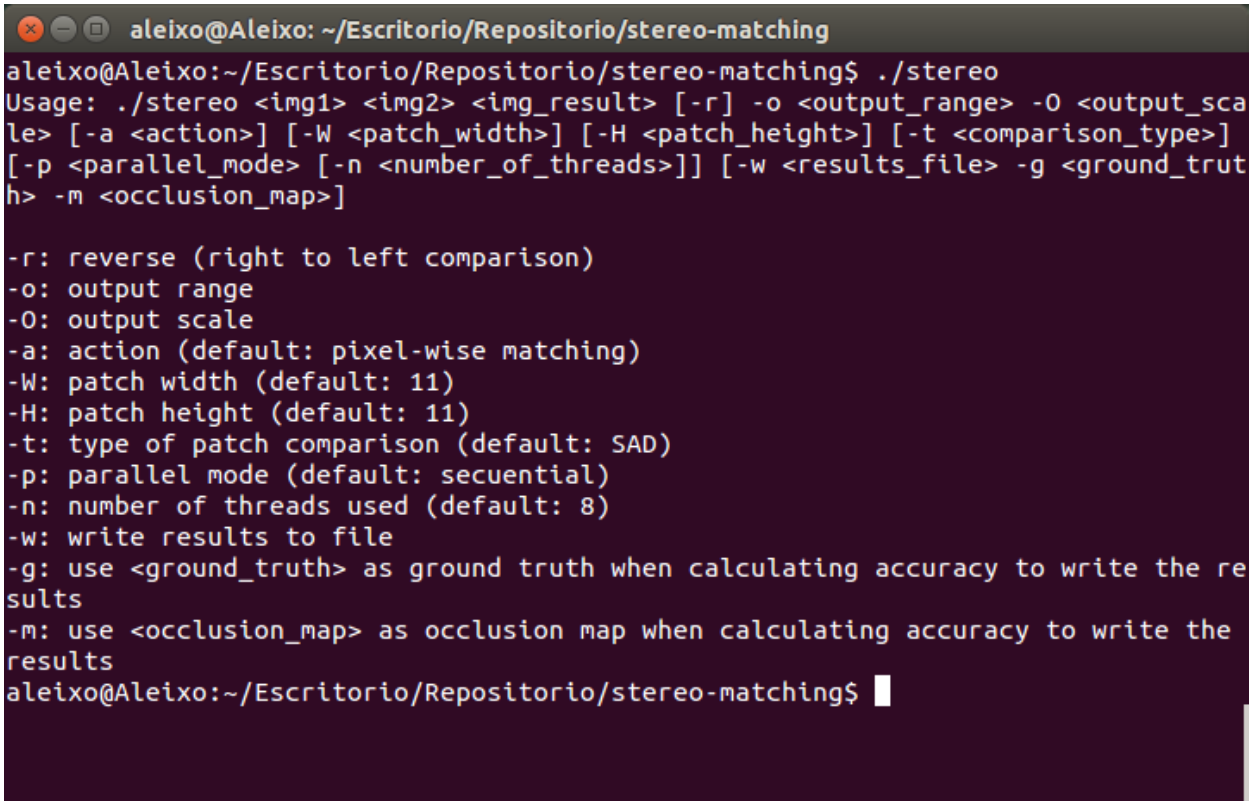


FIGURA 4.6: AYUDA POR LÍNEA DE COMANDOS

Este componente se ha añadido pensando en aportar la posibilidad de desarrollar un front-end completamente independiente del programa, que pueda usar cualquier tipo de tecnología que permita hacer una llamada al sistema operativo para lanzar un archivo ejecutable con determinados parámetros. En este caso, como ya se ha explicado, se ha aprovechado mediante una GUI basada en Java Swing.

En cualquier caso, aunque no se hubiese desarrollado un front-end, esto permitiría la entrada de parámetros en tiempo de ejecución, sin tener que compilar todo el código fuente cada vez que se quiere modificar alguno.

4.3. Back-end en C++

El grueso de la funcionalidad de este software se encuentra en este componente, que recibe parámetros de ejecución por medio de la interfaz de línea de comandos. Desde aquí se gestiona la lectura de los mencionados parámetros, la solicitud de carga y guardado de imágenes, la ejecución de las mediciones de rendimiento sobre los algoritmos de Stereo Matching, así como la de estos mismos, y el almacenamiento de los resultados, si procede.

Este componente se ha escrito en C++ para integrar en él los bloques de código de procesamiento Stereo Matching y mediciones.

4.4. Mediciones de rendimiento

Este componente se encuentra integrado como parte del código del back-end, y consiste en una estructura que permite realizar medidas de rendimiento sobre la ejecución de los algoritmos objeto de este estudio, caracterizado este por el tiempo necesario para llevar a cabo la computación. Se ha diseñado de modo que permita hacer un número arbitrario de medidas independientes sobre la misma ejecución, definido por una constante en tiempo de compilación. De este modo, se puede medir en una sola prueba el rendimiento de varias secciones del algoritmo, lo que resulta muy útil para comparar por secciones las ganancias obtenidas a través del proceso de paralelización y optimización. Por lo tanto, los resultados aportados por este componente para una ejecución del programa consisten en un vector de tiempos de ejecución por áreas de código.

4.5. Algoritmos de Stereo Matching

Este componente es el más importante de la arquitectura, ya que contiene el objeto de este estudio. Está constituido por las diferentes versiones de los algoritmos de Stereo Matching, incluidas simultáneamente como diferentes funciones en un mismo archivo de código fuente para que el back-end pueda llamar a una u otra en base a los parámetros que haya recibido como entrada.

Este bloque es el determinante para la elección de la tecnología de implementación del back-end, ya que tiene unos requerimientos de rendimiento muy importantes, por lo que se ha decidido que su programación debe realizarse en C++, más complejo que otros lenguajes modernos, pero con un gran nivel de control sobre las operaciones que la máquina lleva a cabo y gran eficiencia demostrada en incontables benchmarks comparativos.

Estos algoritmos serán explicados en detalle en el capítulo 5.

4.6. Librería de tratado de imágenes

Para evitar la pérdida de tiempo en el desarrollo de software para manipular imágenes en el formato que se ha elegido para este estudio (PGM) y centrarnos en la

construcción de los componentes dentro del ámbito de interés definido por el proyecto, se ha decidido delegar esta funcionalidad en una librería externa. Todas las llamadas para cargar y guardar imágenes se realizan, por lo tanto, a través de este componente, que ofrece una interfaz sencilla para esto, así como macros que permiten acceder de forma simple a los contenidos como matrices de píxeles.

Esta librería se ha obtenido en forma de código fuente, que es compilado con el programa desarrollado por el alumno.

4.7. Sistema de archivos

Este componente consiste en el propio sistema de archivos del sistema operativo, en el cual se encuentran las imágenes de entrada para los algoritmos, y al cuál se debe acceder para la escritura de los mapas de profundidad generados, así como los resultados de rendimiento si procede. Todos los accesos relacionados con imágenes se realizan, como ya se ha dicho, a través de la librería externa, mientras que los archivos de texto se generan directamente desde el back-end.

5. ALGORITMOS ESTUDIADOS

En esta parte se identificarán los algoritmos que se han elegido para este proyecto de fin de grado, y se realizará un análisis de sus propiedades, así como una explicación sobre su funcionamiento. Cabe recordar que estos algoritmos solamente tratan el emparejamiento de píxeles entre dos imágenes y la construcción de mapas de profundidades relativas. Como ya se ha explicado con anterioridad, el cálculo de distancias reales está fuera del alcance definido para este proyecto. También es importante tener en cuenta el hecho de que se trabaja con imágenes pre-procesadas en las que las líneas epipolares coinciden con las filas de píxeles y para las cuales se dispone de un rango de búsqueda calculado experimentalmente.

Los algoritmos seleccionados son el *Pixel-wise Matching* [2], el *Patch-wise Matching* [3] en dos de sus variantes, y el *Semi-Global Matching* [2].

5.1. Pixel-wise Matching

Es el algoritmo más simple de Stereo Matching. Se ha elegido como método de introducción al desarrollo de software en este campo. A pesar de que su precisión es muy limitada y nunca se utiliza de forma directa, sí puede servir como base para la implementación de algoritmos o para aprovechar su salida como punto de partida para un proceso que optimice el resultado, como en el caso del *Semi-Global Matching*.

El funcionamiento del algoritmo consiste en los siguientes pasos (suponiendo una comparación de izquierda a derecha, para la inversa simplemente se cambia el orden):

1. Para cada píxel de la imagen base:
    a. Se compara con la misma posición de la otra imagen, así como las que estén dentro del rango de búsqueda hacia la izquierda, en base a sus valores de intensidad (dado que la cámara se ha movido hacia la derecha, si el píxel con el que se corresponde no está ocluido, deberá estar en la línea epipolar hacia la izquierda de esas coordenadas).
    b. Se elige el primer píxel con la intensidad más similar al original como mejor candidato de emparejamiento.
    c. Se almacena la distancia entre las coordenadas del píxel original y las del mejor candidato en una matriz de resultados.

2. Para cada píxel de la matriz de resultados:

a. Se multiplica su valor de disparidad por un factor de escala para cubrir el mayor rango de valores posibles teniendo en cuenta que este resultado será almacenado como una imagen con 256 tonos de gris.

Aplicados los pasos anteriores, tenemos como resultado una imagen en escala de grises que representa el mapa de profundidades relativas calculado para la escena, donde el gris más claro (mayores valores de disparidad) representa los puntos más cercanos a la cámara, y el gris más oscuro, los más lejanos.

Podemos ver a continuación una imagen que ilustra este proceso en la Figura 5.1:

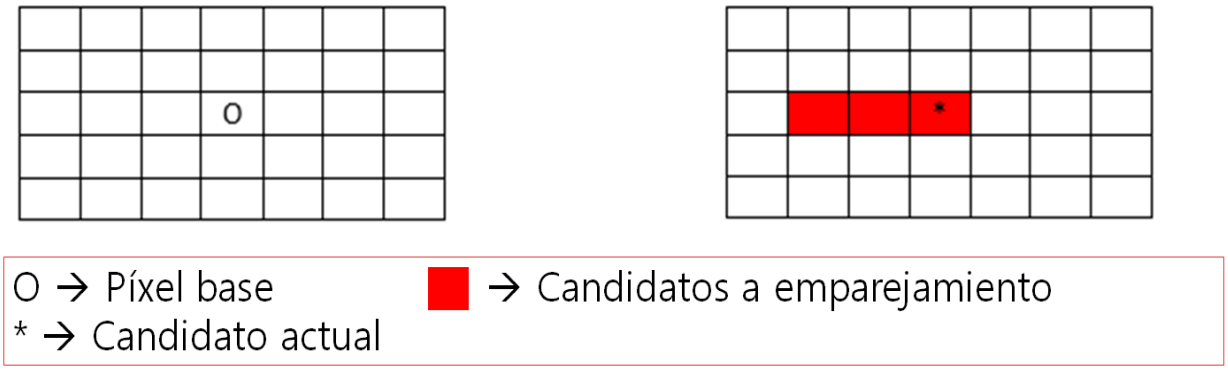


FIGURA 5.1: ILUSTRACIÓN PIXEL-WISE MATCHING

Como se puede ver, el orden de complejidad es muy reducido: simplemente se recorre la imagen haciendo un número fijo de comparaciones por cada píxel, determinado por el rango de búsqueda. Su principal desventaja es su falta de precisión, debida a la ambigüedad en el uso de un único valor de intensidad en una escala tan limitada y sensible a cambios en la iluminación, por ejemplo. Sin embargo, este es un algoritmo muy veloz y su implementación es muy sencilla.

En la Figura 5.2 se incluye un ejemplo de resultado obtenido por medio de este algoritmo para la vista izquierda de un par de imágenes dadas, junto al mapa de profundidad utilizado como *ground truth* (resultado perfecto):

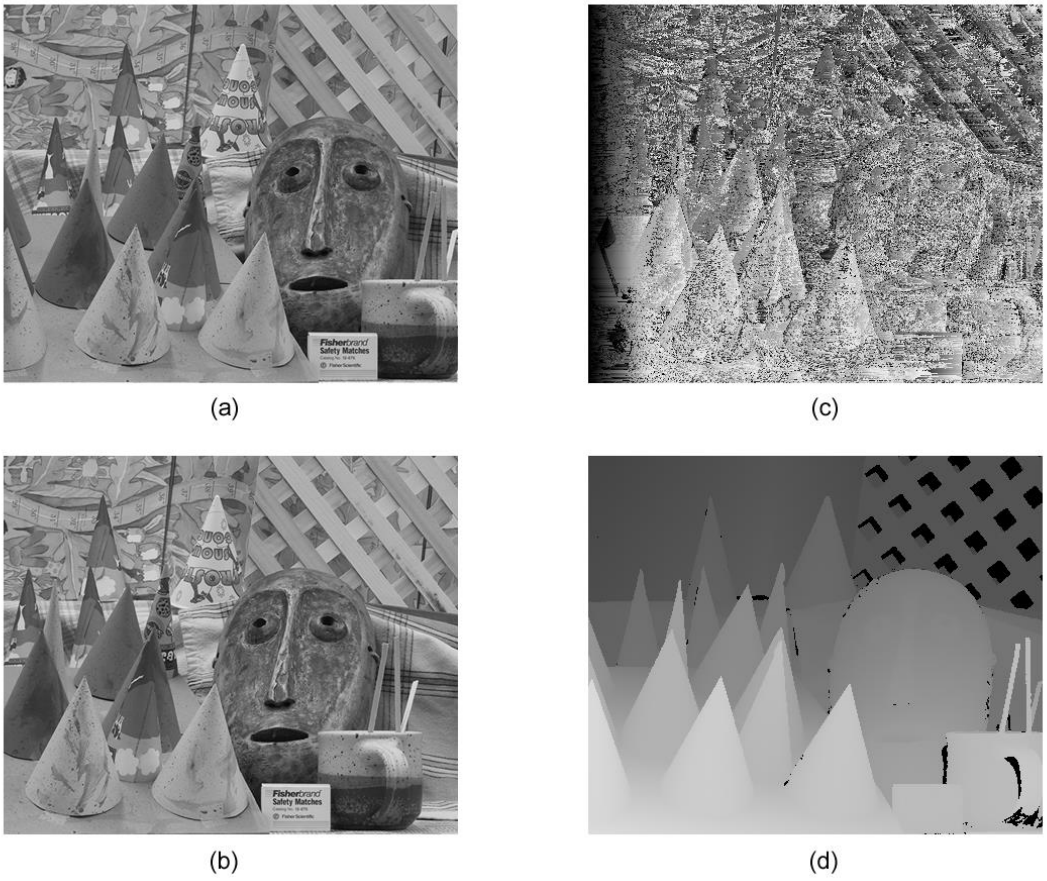


FIGURA 5.2: RESULTADOS PARA PIXEL-WISE MATCHING: (A) IMAGEN IZQUIERDA, (B) IMAGEN DERECHA, (C) MAPA DE PROFUNDIDAD CALCULADO, (D) *GROUND TRUTH*

Como ya se había mencionado, los resultados obtenidos son muy pobres, con un 82.4556% de píxeles erróneos, sin contar los ocluidos, según el mapa de profundidad tomado como correcto, con una tolerancia igual al factor de escala.

5.2. Patch-wise Matching

Este algoritmo representa una evolución directa del presentado en el punto anterior, como se puede ver en el siguiente desglose por pasos de su funcionamiento:

1. Para cada píxel de la imagen base:
    a. Se establece a su alrededor un parche rectangular de un tamaño prefijado. Para valores fuera de la imagen, se establece un valor de intensidad predeterminado.
    b. Se eligen los píxeles candidatos a emparejamiento en base al mismo criterio que el algoritmo anterior.
    c. Se compara este parche con otros del mismo tamaño establecidos alrededor de cada uno de los píxeles candidatos. Para esta comparación pueden seguirse

diferentes estrategias, que dan lugar a variaciones de este algoritmo de entre las cuales dos se contemplan en este estudio y se explicarán a continuación.

d. Se elige como mejor candidato aquél cuyo parche haya tenido una mayor afinidad con el del píxel base, y se toma su separación en coordenadas respecto a las del original como medida de disparidad, agregándola a la matriz de resultados.

2. Para cada píxel de la matriz de resultados:

   a. Se multiplica su valor de disparidad por un factor de escala para cubrir el mayor rango de valores posibles teniendo en cuenta que este resultado será almacenado como una imagen con 256 tonos de gris.

En la imagen de la Figura 5.3 se puede ver una ilustración de este proceso:

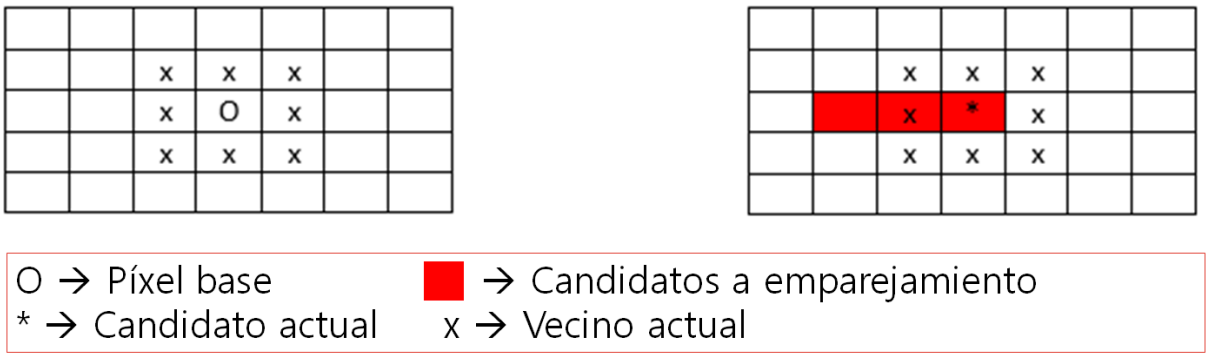


FIGURA 5.3: ILUSTRACIÓN PATCH-WISE MATCHING

Este método proporciona una mayor precisión que el *Pixel-wise Matching* debido a que hay más información disponible acerca de cada uno de los candidatos, lo que puede reducir en gran medida la ambigüedad para puntos parecidos. Esta mejoría depende fuertemente del sistema de comparación de parches utilizados.

Por otra parte, hay que tener en cuenta que, en el tiempo que el algoritmo *Pixel-wise Matching* usaba para hacer una comparación entre dos píxeles, en este deben compararse dos parches, lo que multiplica la complejidad por un factor dependiente del tamaño del propio parche, respecto a la del algoritmo anterior.

5.2.1. Sum of Absolute Differences

Sum of Absolute Differences [3] es la primera de las variaciones del *Patch-wise Matching* comprendidas en el alcance de este proyecto. Implica que para la comparación de dos parches se calcula la diferencia absoluta entre pares de píxeles en la misma posición (en coordenadas relativas dentro del rectángulo) y se toma la suma como coste de emparejamiento para el candidato. Además, se restringe el valor máximo de esta disparidad a una constante prefijada, de modo que dos píxeles claramente pertenecientes a zonas diferentes tengan siempre la misma diferencia entre sí. Esto supone que por cada comparación con un candidato, se han de realizar $W * H + 1$ operaciones aritméticas básicas, donde W es el ancho de la imagen, y H, el alto.

Por otra parte existe un efecto de distorsión al seleccionar parches demasiado grandes con este método, debido a que se pueden estar incluyendo píxeles de diferentes objetos, con lo que los bordes se verán deformados en el mapa de profundidad.

Es necesario, por lo tanto, buscar un balance adecuado en el tamaño de los parches, atendiendo a que:

- Si son demasiado pequeños, no se aportará mucha información adicional al considerar un candidato por parte de su vecindario, por ser este muy reducido, con lo que la ambigüedad del criterio aumenta.

- Si son demasiado grandes, los bordes estarán difuminados por el efecto antes mencionado. Además, el tiempo de procesamiento depende del tamaño del parche, con lo que al algoritmo se volverá más lento.

También se puede utilizar este método en combinación con algoritmos de detección de bordes para reducir el efecto mencionado, aunque esto está fuera del alcance de este proyecto y no se ha probado.

En la figura 5.4 se incluye un ejemplo de resultado obtenido por medio de este método para la vista izquierda de un par de imágenes dadas, junto al mapa de profundidad utilizado como *ground truth* para un tamaño de ventana de 11x11:

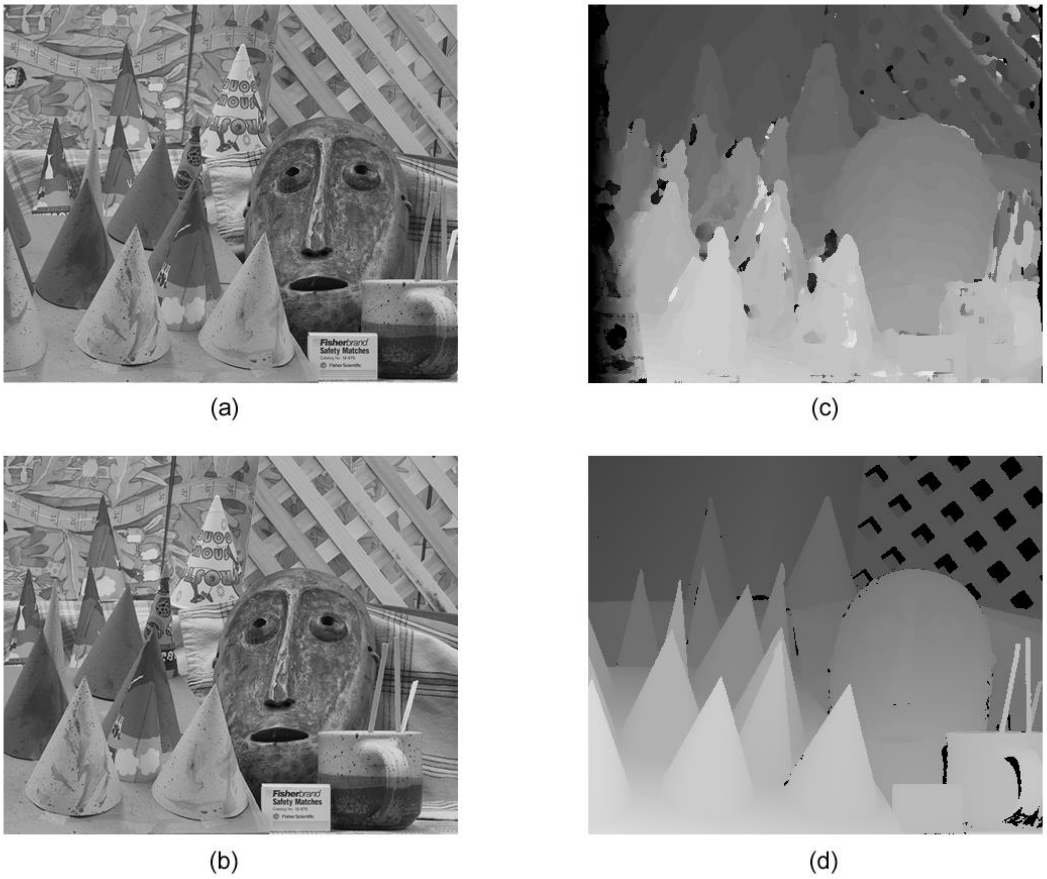


FIGURA 5.4: RESULTADOS PARA SAD CON VENTANA DE 11X11: (A) IMAGEN IZQUIERDA, (B) IMAGEN DERECHA, (C) MAPA DE PROFUNDIDAD CALCULADO, (D) *GROUND TRUTH*

El porcentaje de píxeles erróneos para este resultado es 13.4875%. En la figura 5.5 se adjuntan también, a modo de comparativa, los resultados para un tamaño de parche de 35x35. En ella se puede apreciar el efecto de desvanecimiento de los bordes antes mencionado, con un total de píxeles incorrectos de 16.1917%, pero un resultado claramente menos nítido que el anterior.

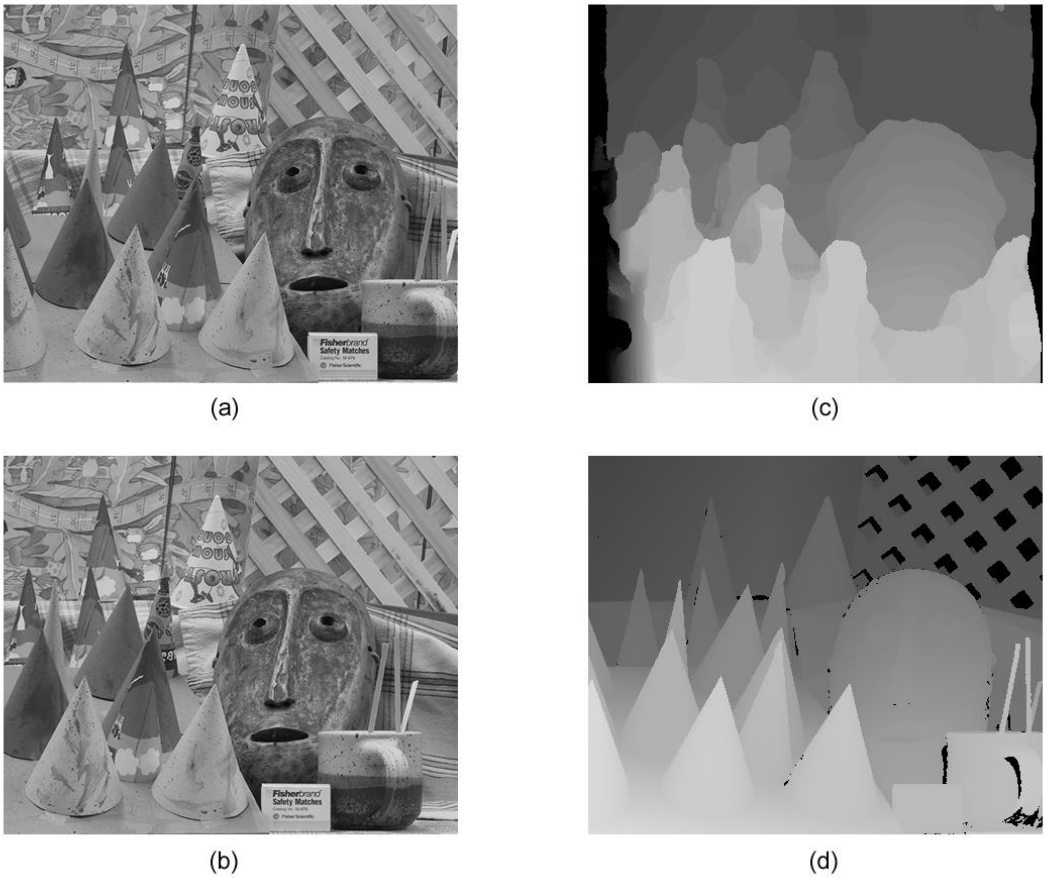


FIGURA 5.5: RESULTADOS PARA SAD CON VENTANA DE 35X35: (A) IMAGEN IZQUIERDA, (B) IMAGEN DERECHA, (C) MAPA DE PROFUNDIDAD CALCULADO, (D) *GROUND TRUTH*

5.2.2. Adaptive Support-Weight

Adaptive Support-Weight [16] es una variación del *Patch-wise Matching* que representa una mejora del método descrito en el punto anterior. El fundamento de la comparación entre los parches es el mismo, pero se añade un factor corrector: las diferencias absolutas son ponderadas en base a la distancia y diferencia de intensidad con el píxel del centro (que será el píxel base o el candidato que se está considerando, para cada uno). Esta ponderación se lleva a cabo aplicando la siguiente ecuación:

$$w(p,q) = \exp\left(-\left(\frac{\Delta C_{pq}}{\gamma_c} + \frac{\Delta g_{pq}}{\gamma_p}\right)\right)$$

ECUACIÓN 5.1: PONDERACIÓN PARA ADAPTIVE SUPPORT-WEIGHT

Donde $w(p,q)$ es el peso asignado al píxel p, perteneciente al vecindario del píxel q, que sería el centro del parche; ΔC_{pq} es la diferencia absoluta entre los valores de intensidad de ambos píxeles, mientras que Δg_{pq}es la distancia euclídea entre ellos y γ_c y γ_p, constantes fijadas de forma experimental.

Una vez calculada la ponderación anterior se aplica la siguiente ecuación para calcular el coste de emparejamiento del par de píxeles:

$$E(p, p_d) = \frac{\sum_{q \in N_p, q_d \in N_{p_d}} w(p,q) w(p_d, q_d) e(q, q_d)}{\sum_{q \in N_p, q_d \in N_{p_d}} w(p,q) w(p_d, q_d)}$$

ECUACIÓN 5.2: COSTE DE EMPAREJAMIENTO PARA ADAPTIVE SUPPORT-WEIGHT

Donde $E(p, p_d)$ representa el coste de emparejamiento de los píxeles p y p_d (siendo d la disparidad entre el píxel base y el candidato), N_p es el vecindario de p, $w(p, q)$ mantiene el significado de la anterior ecuación y $e(q, q_d)$ es la diferencia absoluta con truncamiento entre los píxeles q y q_d.

Este sistema de ponderaciones consigue diferenciar los píxeles que probablemente no pertenecen al mismo objeto que el candidato o el píxel base de entre los de los respectivos parches para que sus valores no tengan repercusiones relevantes en el resultado final. Con esto se garantiza una mayor robustez para tamaños grandes de ventana que mediante el uso del método descrito en el punto anterior, ya que se elimina el efecto de difuminado de los bordes. No obstante, también se realiza un mayor número de operaciones aritméticas por cada candidatura evaluada, que también resultan más complejas, incluyendo exponenciaciones y raíces cuadradas. Esto lo convierte en una solución más lenta que el *Sum of Absolute Differences*.

Con esto se cambia el paradigma respecto al método anterior en el que, para conseguir una precisión óptima, había que elegir un buen balance en el tamaño del parche. En este caso, como norma general un parche mayor significa mejores resultados, con lo que solamente hay que cuidar el balance entre precisión y rendimiento, lo cual también era importante en el algoritmo anterior.

En la Figura 5.6 se incluye un ejemplo de resultado obtenido por medio de este método para la vista izquierda de un par de imágenes dadas, junto al mapa de profundidad utilizado como *ground truth* para un tamaño de ventana de 11x11:

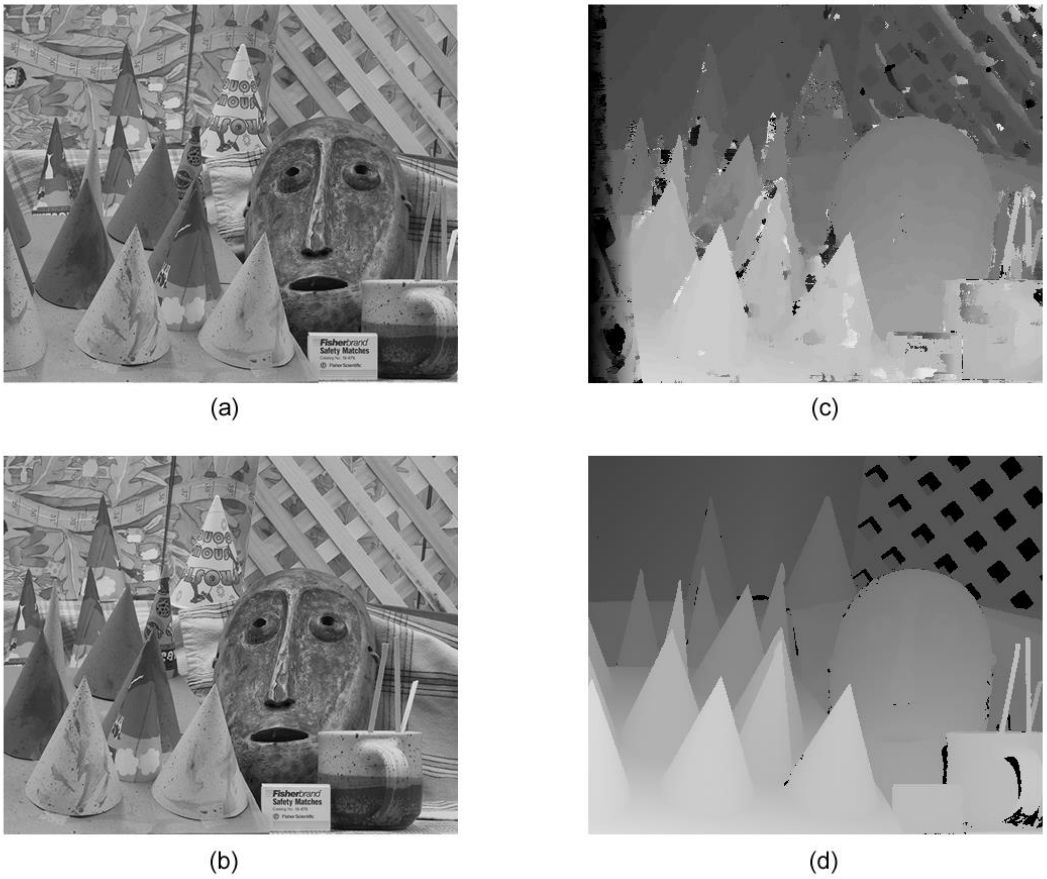


FIGURA 5.6: RESULTADOS PARA ASW CON VENTANA DE 11X11: (A) IMAGEN IZQUIERDA, (B) IMAGEN DERECHA, (C) MAPA DE PROFUNDIDAD CALCULADO, (D) *GROUND TRUTH*

El porcentaje de píxeles erróneos para este resultado es 13.5035%, algo peor que el método anterior para el mismo tamaño de ventana. En la figura 5.7 se adjuntan también, a modo de comparativa, los resultados para un tamaño de parche de 35x35.

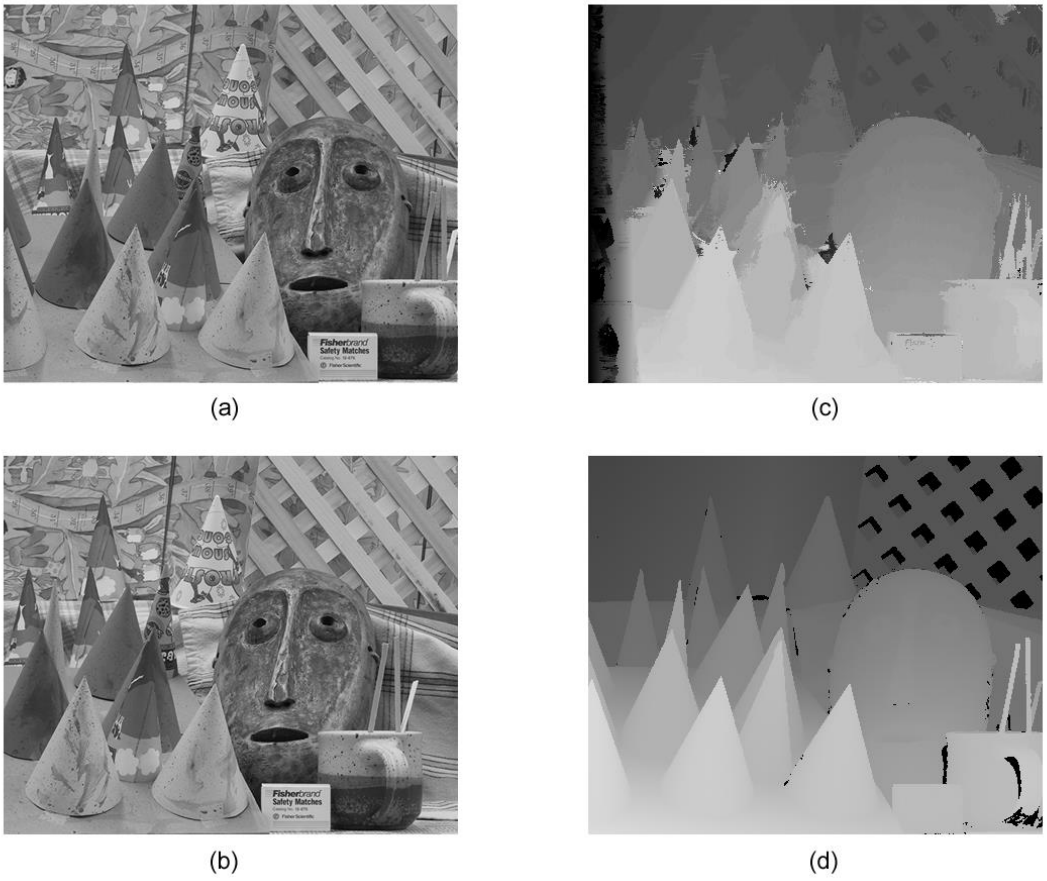


FIGURA 5.7: RESULTADOS PARA ASW CON VENTANA DE 35X35: (A) IMAGEN IZQUIERDA, (B) IMAGEN DERECHA, (C) MAPA DE PROFUNDIDAD CALCULADO, (D) *GROUND TRUTH*

Podemos observar que no se da el efecto del que adolecía el anterior método para tamaños de ventana más grandes, conservando de forma perfectamente nítida la definición de los distintos objetos de la escena. De hecho, la precisión se ha mejorado respecto al parche de 11x11, con un total de píxeles erróneos del 8.20352%, de acuerdo con lo que se había dicho antes.

5.3. Semi-Global Matching

Esta aproximación al problema del Stereo Matching cambia radicalmente el enfoque respecto a los métodos vistos anteriormente: en lugar de emplear el grueso del cálculo en la búsqueda directa del candidato más adecuado para cada píxel, toma como entrada la solución de un algoritmo que se sabe impreciso pero muy rápido (en este caso se ha utilizado el *Pixel-wise Matching*), y la transforma mediante la imposición de una restricción de suavidad para obtener resultados con un alto nivel de exactitud.

Dicha restricción de suavidad consiste en la penalización de diferencias de disparidad entre píxeles contiguos mediante el coste de emparejamiento en un proceso iterativo. Es decir, se asume que píxeles contiguos pertenecen a la misma

superficie y se encarece intencionadamente el coste de los emparejamientos que no respeten esto. Esta restricción se aplica, por lo tanto, direccionalmente. A continuación se describen los pasos del algoritmo:

1. Se calcula una solución inicial mediante un método computacionalmente poco costoso. En este estudio hemos utilizado una variante del *Pixel-wise Matching* que, en lugar de seleccionar un mejor candidato, guarda en una matriz tridimensional los costes para todos ellos para devolverla como resultado. Esta matriz sería, por lo tanto, de tamaño $W * H * R$, donde W es el ancho de la imagen, H el alto y R el rango de búsqueda.

2. Partiendo de esta matriz, se elige un número arbitrario de pares dirección sentido sobre los que aplicar la restricción de suavidad. Experimentalmente se ha determinado como número recomendado cualquiera entre 8 y 16. En este caso hemos elegido 8 para simplificar la codificación, ya que cada píxel está rodeado por 8.

3. Se define una matriz de reducción del mismo tamaño que la que contiene los costes iniciales para acumular los costes de emparejamiento para cada disparidad de cada píxel calculados en las iteraciones correspondientes a estas direcciones.

4. Para los vectores correspondientes a las direcciones y sentidos seleccionados:
   a. Para las disparidades de cada píxel se calcula el coste de la restricción de suavidad de acuerdo a la siguiente fórmula:

$$L_r(p,d) = C(p,d) + \min\begin{pmatrix} L_r(p-r,d), L_r(p-r,d-1)+P_1, \\ L_r(p-r,d+1)+P_1, min_i L_r(p-r,i)+P_2 \end{pmatrix} - min_i L_r(p-r,i)$$

ECUACIÓN 5.3: RESTRICCIÓN DIRECCIONAL DE SUAVIDAD PARA SEMI-GLOBAL MATCHING

Donde $L_r(p,d)$ es el coste de la restricción de la suavidad en la dirección y sentido correspondientes al vector r para el píxel p con disparidad d; $C(p,d)$ es el coste pre-calculado en el primer paso; y P_1 y

P_2 las penalizaciones pequeña y grande por discontinuidad de la profundidad, respectivamente. Esta ecuación favorece que píxeles vecinos tengan profundidades similares, ya que para para disparidades similares a la mejor candidata del píxel anterior en la dirección actual se sumará menos coste adicional. La substracción del último término se explica como un controlador para evitar que el coste se vaya acumulando a lo largo de la dirección hasta valores extremos. No afecta a los cálculos sobre el píxel actual, ya que se resta el mismo valor para todas las disparidades.

   b. Para los píxeles que no tengan ningún predecesor en la dirección elegida, se toma directamente el coste pre-calculado en la fase inicial como coste de la restricción de suavidad de la dirección actual. Esto se aplica para todas las disparidades posibles del píxel.
   c. El cálculo realizado para cada disparidad de cada píxel se agrega a la matriz de resultados en la posición correspondiente.

5. Con la matriz de reducción resultante de haber aplicado la restricción de suavidad en todas las direcciones y sentidos elegidos, para cada píxel se elige como mejor candidata la disparidad que tenga un menor coste acumulado, agregándola a la matriz de resultados.

6. Para cada píxel de la matriz de resultados:
   a. Se multiplica su valor de disparidad por un factor de escala para cubrir el mayor rango de valores posibles teniendo en cuenta que este resultado será almacenado como una imagen con 256 tonos de gris.

En base a lo expuesto, podemos observar que la cantidad de cálculos necesarios por candidato no es muy elevada, con un coste computacional para los valores iniciales despreciable y muy liviano para la aplicación de la constante de suavidad a lo largo de las direcciones elegidas, dependiente de estas últimas.

Por otra parte, el tener en cuenta los diferentes objetos de la escena como superficies con la misma profundidad, dejando un ligero margen para superficies curvas (la penalización para la diferencia unitaria de profundidad entre píxeles vecinos es mucho menor que para disparidades mayores), garantiza un buen grado de

coherencia en la asignación de profundidades a los distintos puntos de la imagen, especialmente en escenarios con pocos objetos que estén bien definidos.

En la Figura 5.8 se incluye un ejemplo de resultado obtenido por medio de este algoritmo para la vista izquierda de un par de imágenes dadas, junto al mapa de profundidad utilizado como *ground truth*:

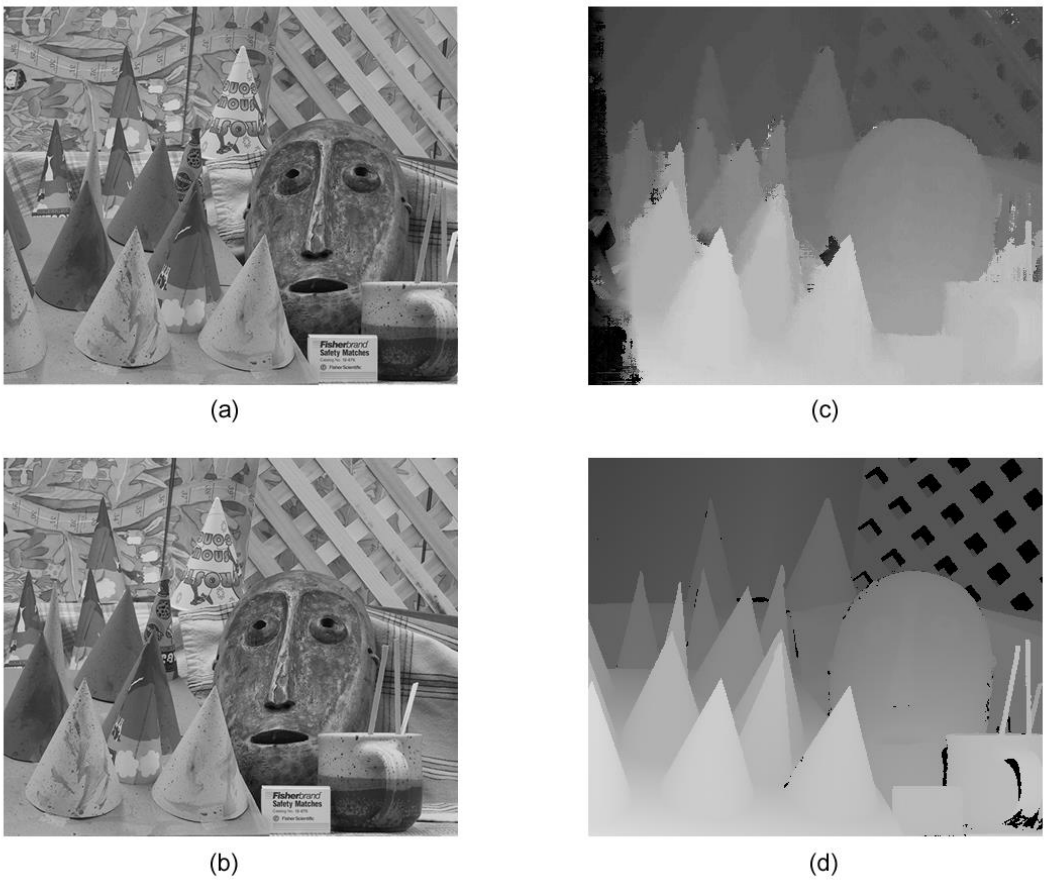


FIGURA 5.8: RESULTADOS PARA SEMI-GLOBAL MATCHING: (A) IMAGEN IZQUIERDA, (B) IMAGEN DERECHA, (C) MAPA DE PROFUNDIDAD CALCULADO, (D) *GROUND TRUTH*

Se puede observar que se han respetado las superficies de los objetos, cuyos valores de profundidad son muy homogéneos. También es reseñable que, a pesar de haber partido del resultado de la aplicación del algoritmo *Pixel-wise Matching* como solución inicial, que ofrecía un resultado de 82.4556% de píxeles mal calculados para este mismo par de imágenes, se ha obtenido un 7.95965% de píxeles incorrectos por medio de la aplicación de la restricción de suavidad, menos de un 10% de los del resultado original.

6. ARQUITECTURAS HARDWARE CONSIDERADAS

En este capítulo se tratarán en profundidad las arquitecturas paralelas consideradas para la realización de este estudio. Como ya se ha dicho en la Introducción, estas son:

- Intel Xeon E5-2630Lv2 [5]
- Intel Xeon Phi 7120P [6]
- Placa *Parallella* [7]

Como ya se ha mencionado previamente, no se han podido realizar mediciones sobre el microprocesador portátil, debido a la falta de documentación sobre su uso y a dificultades técnicas siguiendo la guía de instalación. Se ha conseguido que corra el sistema operativo diseñado *ad hoc* para ella a pesar de desconocer la versión exacta del hardware, tanto la versión para uso con monitor como la más básica, que proporciona únicamente la posibilidad de acceso mediante SSH en LAN. No obstante, en la primera no se ha conseguido generar un output de vídeo a un monitor, mientras que la segunda, a pesar de responder a las peticiones SSH, no admite el usuario por defecto según la documentación. Dada la estimación hecha de los recursos que necesitaría resolver estos problemas técnicos, se ha decidido prescindir de este estudio.

Por los motivos anteriores, desde este punto centraremos la atención del trabajo en los otros dos sistemas. Ambas arquitecturas han sido puestas a disposición del alumno a través de una estación de trabajo con las siguientes características:

- Servidor Dell PowerEdge R720
- Dos procesadores Intel Xeon E5-2630Lv2
- 32 GB de memoria RAM
- Una tarjeta Intel Xeon Phi 7120P
- Sistema operativo Centos 6.6
- Intel® Many-core Platform Software Stack (MPSS) 3.5.1.
- Intel Parallel Studio XE 2015 update2

A continuación explicamos en detalle ambas arquitecturas y cómo se ha enfocado el trabajo con cada una.

6.1. Intel Xeon E5-2630Lv2

Esta arquitectura cumple la función de procesador principal del servidor y, como tal, el modo de trabajo empleado para realizar las mediciones sobre ella es el mismo que sobre cualquier otra plataforma estándar: se compila el código en la propia

máquina y se ejecuta. Para esto, se ha utilizado el compilador de Intel ICC, ya que favorece el rendimiento al utilizar sus procesadores de servidor.

Cada procesador de este tipo tiene seis núcleos físicos. Como podemos ver, el nivel de paralelismo es ligeramente más alto que el habitual para ordenadores personales contemporáneos, pero aun así está lejos de los estándares de supercomputación paralela. No obstante, como ya se ha mencionado, se trabaja con un *cluster* que agrupa dos de estos procesadores, con lo que el número de núcleos físicos utilizables por el software es de 12.

Hay que tener en cuenta que a la hora del reparto de hilos entre ambos procesadores ya se encarga el sistema operativo (*scheduling*) de forma automática, de modo que el programador no interviene en esta distribución.

Cabe esperar que cada uno de los núcleos sea más potente que aquellos de un ordenador personal de la misma generación y que, por lo tanto, esta arquitectura funcione bien para un número de hilos limitado, del orden del número de núcleos físicos.

Como ya se ha comentado, el alumno no dispone de acceso directo al servidor físico, de modo que el trabajo con este *cluster* se realiza a través de un cliente SSH sobre una red VPN o desde la estación de trabajo que tiene asignada en el CITIUS. Además, las restricciones sobre la conexión a Internet de este servidor obligan a copiar manualmente los archivos del proyecto a través de la herramienta SCP.

6.2. Intel Xeon Phi 7120P

Esta arquitectura *many*-core supone un cambio de paradigma frente a la anterior, pues su función es la de coprocesador de un servidor. Es decir, este sistema está pensado para que el procesador principal descargue en él el trabajo que sea altamente paralelizable para aprovechar el elevado número de núcleos físicos de que dispone (61). Estos núcleos, además, cuentan con tecnología Intel Hyper-Threading [17], que permite la ejecución paralela de varios hilos (en este caso, 4) en un solo núcleo, consiguiendo un aumento de la eficiencia a través de una planificación de los cambios de contexto que trata de evitar la espera prolongada de algunas instrucciones. Por esto, el número de núcleos lógicos asciende a la cifra de 244, valor que ya representa un nivel de paralelismo dentro del ámbito de la supercomputación.

Por otra parte, esta máquina corre una versión reducida de Linux que permite el acceso a ella a través de SSH desde el procesador principal. De este modo, para utilizar este coprocesador, el alumno accede por medio de un cliente SSH sobre una red VPN al cluster de Xeon, y desde ahí, utilizando otra vez esta herramienta de trabajo remoto se autentica en el Xeon Phi.

En cualquier caso, la versión reducida de Linux que corre este coprocesador no incluye software para la compilación, ya que la ejecución de código en esta arquitectura está diseñada para hacerse de uno de los dos siguientes modos:

- **Ejecución nativa**: Como ya se ha dicho, el coprocesador carece de herramientas de compilación propias, por lo que para realizar una ejecución nativa, es necesario realizar una compilación cruzada del código, desde el procesador principal, por ejemplo. Esto significa que se correrá en la otra arquitectura un programa que generará código máquina adaptado para esta. El compilador de Intel ICC facilita enormemente esta labor, ya que permite llevar a cabo la compilación cruzada con la simple adición de un *flag* extra. Una vez compilado, el ejecutable debe transferirse al sistema de archivos manejado por el coprocesador a través de SCP, y el usuario debe autenticarse en este sistema con una cuenta propia para ejecutar el programa.
- **Ejecución parcial como *offload*** [18]: Otra modalidad, pensada para la cooperación directa entre procesador principal y coprocesador, pasa por la compilación del programa de forma directa para el procesador principal, con la declaración de secciones de *offload* que se descargan sobre el coprocesador (muy similar a la forma de trabajar con GPUs). Esta forma de ejecución está pensada para software con partes altamente paralelas bien definidas, que puedan aprovechar mejor la arquitectura many-core del Xeon Phi, sin renunciar a la ejecución más eficiente de las partes secuenciales o con paralelismo reducido sobre el Xeon principal.

Para este proyecto, se ha decidido que el segundo modo de ejecución no forma parte de su alcance, de modo que todas las pruebas que se realicen sobre esta arquitectura se harán en base a una compilación cruzada del código.

Otra característica importante de esta máquina es que permite definir la distribución por parte del usuario de forma manual de los hilos a la hora de correr un software. Existen tres opciones principales [19]:

- **Compact**: La distribución de hilos se realiza rellenando cada núcleo antes de pasar al siguiente. Teniendo en cuenta que la tecnología Intel Hyper-Threading permite el lanzamiento de cuatro hilos de forma semi-simultánea en el mismo núcleo, la distribución sería:

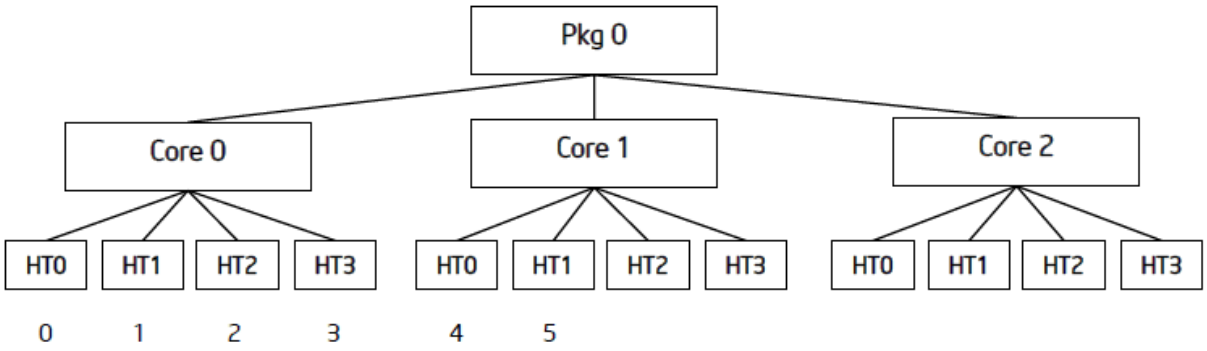


FIGURA 6.1: INTEL THREAD AFFINITY: MODO COMPACT

- **Scattered**: En este caso, los hilos se distribuyen asignando uno a cada núcleo hasta completar la primera capa en todos, en cuyo caso se volvería a comenzar desde el primero para la siguiente. Esto se puede ver en la Figura 6.2:

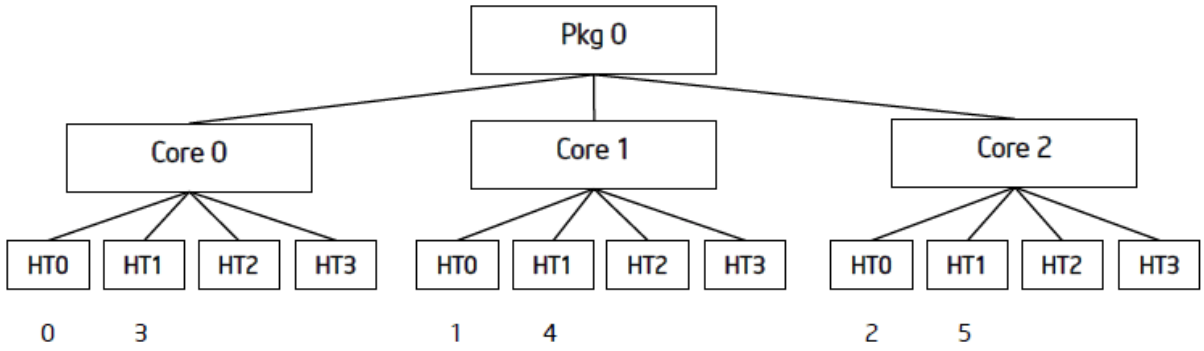


FIGURA 6.2: INTEL THREAD AFFINITY: MODO SCATTERED

- **Balanced**: Este modo es muy similar al anterior, con la diferencia de que se agrupan en el mismo núcleo hilos con números consecutivos. Esto se ilustra en la figura 6.3:

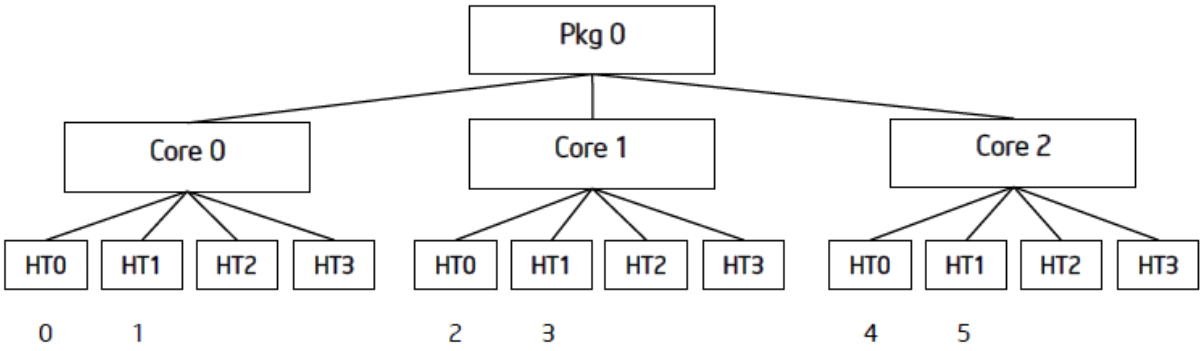


FIGURA 6.3: INTEL THREAD AFFINITY: MODO BALANCED

El modo de reparto de hilos se puede elegir en tiempo de compilación, ya sea mediante una instrucción explícita para ello en el código fuente o bien utilizando una variable de entorno de la consola, siguiendo el estilo de las preferencias en OpenMP.

Estas opciones abren nuevas posibilidades a la hora de analizar el rendimiento del software en esta arquitectura, que serán revisadas en el apartado correspondiente al análisis de resultados.

7. IMPLEMENTACIÓN PARALELA Y OPTIMIZACIÓN

En este capítulo trataremos la implementación de los algoritmos de Stereo Matching considerados y, más en profundidad, las mejoras de rendimiento que se han aplicado a cada uno a través de técnicas de paralelización y optimización.

Como ya se ha dicho en el capítulo dedicado a la arquitectura del software desarrollado, cada uno de estos algoritmos ha sido implementado usando el lenguaje C++ como una función, y son llamados por el programa principal en función de los parámetros recibidos por línea de comandos. Esto se aplica también a las versiones paralelas, incluidas en otras funciones como si de algoritmos distintos se tratase.

Las versiones básicas de los algoritmos se corresponden con la implementación directa de los pseudo-códigos mostrados en el apartado anterior. El proceso de desarrollo de las versiones paralelas se explica a continuación.

7.1. Pixel-wise Matching y Patch-wise Matching

Se ha creado un único apartado para estos dos algoritmos debido a que su paralelización es similar y se puede establecer una analogía muy directa entre ambas.

Como ya hemos visto en el capítulo anterior, el *Pixel-wise Matching* recorre píxel a pixel la imagen buscando un candidato para el emparejamiento en las líneas correspondientes de la otra imagen. No existen, pues, dependencias entre el cálculo de profundidad para dos píxeles diferentes.

Lo mismo sucede con el *Patch-wise Matching*, cuya única diferencia es que utiliza una ventana de vecinos para reducir la ambigüedad en la elección del mejor candidato.

De este modo, se pueden paralelizar los dos algoritmos asignando a cada hilo una cantidad determinada de filas o columnas de la imagen para que las procese independientemente del resto. Así, se almacenarán los resultados en una matriz de píxeles común, en cuya reducción no se darán condiciones de carrera debido al hecho de que el resultado para cada píxel se almacena en posiciones diferentes de dicha matriz. Con la cantidad de filas y columnas que tiene una imagen, está garantizado un reparto balanceado para cantidades de hilos incluso elevadas, con lo que no sería necesario recurrir a un refinamiento mediante paralelización anidada u otras estrategias.

7.2. Semi-Global Matching

La paralelización del algoritmo *Semi-Global Matching* tiene una mayor complejidad que la de los algoritmos anteriores. A efectos de su paralelización, este algoritmo puede dividirse en tres bloques claramente definidos, ya que las estrategias

para abordarlos serán diferentes entre sí. Dichos bloques son los que se listan a continuación:

- **Cálculo de la solución inicial (Bloque 1)**: Como ya hemos explicado, este se realiza mediante un método basado en el *Pixel-wise Matching*, con la diferencia de que, en este caso, en lugar de almacenar directamente el mapa de profundidad, se mantienen los costes de emparejamiento para cada candidato respecto a cada píxel base. En cualquier caso, la metodología de paralelización es la misma que en el algoritmo original.
- **Ejecución del núcleo del algoritmo (Bloque 2)**: Esta fase engloba el cálculo de las soluciones parciales derivadas de la extensión de la restricción de suavidad en cada uno de los pares dirección-sentido elegidos, así como su reducción en una matriz común. Es la parte con mayor complejidad a nivel de posibilidades de paralelismo y, por lo tanto, se tratará de forma separada más adelante, desglosada por metodologías en los correspondientes sub-apartados.
- **Selección de la profundidad para cada píxel (Bloque 3)**: Dada la matriz de salida del bloque anterior, en este se recorrerá buscando para cada píxel la disparidad que menor coste acumulativo tenga. Simplemente se trata de recorrer una matriz tridimensional sin ningún tipo de dependencias en la consideración de distintos píxeles. De este modo, la paralelización más directa consiste en repartir una de las dimensiones de la matriz entre el número de hilos. Las dimensiones correspondientes al ancho y al alto de la imagen parecen las más adecuadas, pues sus valores son más elevados y permiten un reparto equitativo de la carga de trabajo entre un mayor número de hilos.

La complejidad de paralelización del segundo bloque se debe a las dependencias de datos que presenta. Por un lado, la extensión de la restricción de suavidad a lo largo de cada par dirección-sentido obtiene un resultado que debe combinarse con el resto de soluciones parciales para la consecución de solución total. Esto supone una dependencia básica de reducción. Por otra parte, el cálculo del coste de emparejamiento de una determinada disparidad para un determinado píxel para cada par dirección sentido depende del anterior, de modo que la división de la imagen para el reparto de la carga de trabajo entre los hilos no es inmediata, y depende de la dirección y sentido que estén siendo considerados.

A continuación se explicarán las distintas estrategias seguidas para la paralelización de este algoritmo, diferenciadas entre sí por la aproximación utilizada para el tratamiento del bloque central. Para los otros dos bloques, en todas ellas se utilizan los métodos descritos en los puntos anteriores.

7.2.1. Direcciones

La aproximación más natural para abordar este problema podría ser el cálculo independiente de cada solución parcial vinculada a una dirección y un sentido. Para ello habría que crear una matriz por cada una de estas soluciones parciales (en la versión secuencial solamente hacía falta una en total). Estas matrices serán luego reducidas en una con la solución definitiva. Resulta evidente que no se pueden reducir a la par de la realización de los cálculos parciales, como permite la versión secuencial ya que, en este caso, varios hilos estarán accediendo a las mismas posiciones de memoria de forma simultánea, lo que podría provocar condiciones de carrera.

Como solución para el problema de la citada reducción, se ha considerado la división de la matriz en franjas, tantas como pares dirección-sentido. De este modo, se crea un *mutex* para cada una de estas franjas y cada uno de los hilos comienza la reducción de sus soluciones parciales asociadas en una franja diferente, bloqueándola para evitar que el resto acceda a ella mientras tanto, evitando así carreras críticas. Una vez completada la franja, se pasa a la siguiente en un modelo circular de modo que, debido al balanceo en el reparto de la matriz, se espera que el tiempo perdido respecto a una reducción completamente paralela sea muy reducido.

De lo anterior se puede deducir que este método funcionará bien para cantidades de hilos reducidas pues, como se ha dicho en el capítulo anterior, se ha elegido arbitrariamente ocho como el número de pares dirección-sentido que serán consideradas, de modo que un número de hilos mayor no mejorará el resultado para este bloque (aunque sí podría mejorar los otros dos). Un número elevado de hilos podría incluso reducir el rendimiento, dado que el coste de lanzarlos podría no verse compensado por las ganancias de rendimiento obtenidas en la ejecución de los otros bloques.

7.2.2. Franjas

Otro modo de abordar la paralelización del núcleo del algoritmo es la división de la solución utilizada como punto de partida en lo que denominamos franjas. Como ya hemos dicho, existen dependencias en los cálculos realizados para cada píxel de una solución parcial con el anterior en la dirección y sentido correspondientes. Es necesario en este caso, por lo tanto, especificar una estrategia de paralelización diferente para la obtención de cada resultado parcial. Estas se dividen en dos tipos principales:

- **Direcciones horizontales o verticales respecto a las coordenadas de la imagen**: En este caso el reparto de la matriz es muy directo. Para las direcciones horizontales se reparten las filas en franjas del mismo grosor y para las verticales, las columnas. De este modo, cada uno de los hilos puede avanzar independientemente del resto, con dependencias reducidas al interior de su franja

- **Direcciones diagonales respecto a las coordenadas de la imagen**: En este caso, la división de la matriz de solución parcial en franjas resulta más compleja, dado que la longitud de las diagonales es variable, y no es viable el reparto de la misma entre varios hilos, ya que no se mejoraría el rendimiento de una versión secuencial debido a que el cálculo para cada píxel tendría que esperar al del anterior. Por esto, la solución concebida se basa en el cálculo del mejor reparto posible de diagonales enteras. Con este objetivo, se computa el número de píxeles que correspondería en un reparto equitativo para cada uno de los hilos. Con este dato, se van sumando franjas de forma ordenada sin sobrepasar esta cantidad. Para un reparto más compensado, se redondeará alternamente el número de diagonales enteras que es responsabilidad de cada hilo. Para el cálculo de la cantidad de píxeles que aporta cada diagonal se ha utilizado el número de la propia diagonal hasta llegar a la sección central, existente en imágenes cuyo alto y ancho difieren, en cuyas diagonales el número de píxeles es igual a la menor de estas dimensiones. Después de la sección central, se resta el número de diagonal a la suma de ambas dimensiones. Para un mejor entendimiento, se puede observar el fragmento de código en C de la figura 7.1.

```
diagonals[0] = 0;
diagonals[N_THREADS] = w + h - 1;
for(k=0; k<N_THREADS-1; k++){
        int pCount, d = diagonals[k], aux;
        pCount = 0;
        while(1){
                if(d >= diagonals[N_THREADS]){
                        diagonals[k + 1] = diagonals[N_THREADS];
                        break;
                }

                if(d < minHW) aux = d;
                else if(d < maxHW) aux = minHW;
                else aux = sumaHW - d - 1;

                pCount += aux;
                if(pCount < pixelsPerThread) d++;
                else if(takeExcess) {
                        diagonals[k + 1] = d + 1;
                        takeExcess = !takeExcess;
                        break;
                } else {
                        diagonals[k + 1] = d;
                        takeExcess = !takeExcess;
                        break;
                }
        }
}
```

FIGURA 7.1: CÓDIGO EN C PARA REPARTO DE DIAGONALES

Cabe añadir a lo anterior que solamente es necesario realizar una vez el reparto de las diagonales entre los hilos, dado que este cálculo será extrapolable a los diferentes pares dirección-sentido, debido a la simetría de cualquier rectángulo tomando como eje su diagonal.

También es reseñable que esta aproximación no necesita utilizar más de una matriz para los resultados parciales, ya que estas se computan de forma secuencial entre sí. Esto significa, además, que la reducción de dichas soluciones parciales se puede realizar de forma óptima, a medida que se calculan y sin tener que utilizar estrategias para evitar condiciones de carrera que limiten el rendimiento posible.

Como se puede observar, esta estrategia ofrecerá una mejor escalabilidad que la anterior para cantidades de hilos elevadas, dado que podrá hacer un reparto equitativo de la parte de cálculo más pesada del programa. No obstante, hay que tener

en cuenta el desbalanceo que la restricción de tomar diagonales enteras impone, y que el hecho de que el tamaño de estas no será siempre el mismo, puede desembocar en que su reparto sea menos eficiente que aquél de las filas y columnas llevado a cabo para otras soluciones parciales.

7.2.3. Combinación de direcciones y franjas

Esta estrategia combina las dos presentadas en los puntos anteriores, es decir, lo que se propone es una paralelización anidada con dos niveles: el exterior, que trata los pares dirección-sentido seleccionados; y el interior, que abarca para cada solución parcial la extensión de la restricción de suavidad.

Se seguirán para este método los planteamientos ya expuestos con anterioridad para los otros dos: el cálculo paralelo de las soluciones parciales con reducciones ordenadas controladas por *mutex* por una parte, y la paralelización individual de cada par dirección-sentido por la otra.

Esta solución se propone de cara a la utilización de cantidades de hilos muy elevadas, del orden de las dimensiones de la imagen ya que, para que el rendimiento sea alto, tiene que haber un número de hilos que cause una pérdida de eficiencia en la estrategia de utilizar únicamente franjas, pues en este caso se añade la complejidad de la paralelización de la reducción, que no era necesaria con el enfoque anterior. Se espera, por lo tanto, que para ejecuciones con un nivel de paralelismo reducido esta versión del algoritmo ofrezca peores resultados que la anterior estrategia, pero que escale mejor de cara a sistemas con un número de núcleos muy elevado.

Por los motivos expuestos, esta estrategia se ha desarrollado con un claro enfoque a su ejecución sobre la arquitectura Intel Xeon Phi 7120P [6], que parece reunir las características necesarias para sacarle el máximo provecho.

8. ANÁLISIS DE RENDIMIENTO

En este capítulo se describirán las baterías de pruebas realizadas para obtener los resultados de rendimiento en las diferentes arquitecturas paralelas consideradas. También se realizará un análisis de dichos resultados desde varias perspectivas que se explicarán a lo largo de los siguientes apartados.

Como ya hemos explicado en el apartado referente a las arquitecturas consideradas, se ha descartado la realización de pruebas sobre la placa *Parallella* por motivos técnicos, con lo que nos hemos centrado en el Intel Xeon E5-2630Lv2 y en el Intel Xeon Phi 7120P para llevar a cabo las mediciones. De este modo, el análisis se dividirá en los dos apartados correspondientes, añadiendo otro a continuación para una breve comparativa de ambas arquitecturas.

Cabe añadir que, a pesar de que se han realizado mediciones para cuatro imágenes distintas, los comportamientos de los algoritmos al tratarlas en cuanto a rendimiento son muy similares entre sí, por lo que para este estudio consideraremos solamente una de ellas ("*Cones*") como representativa del resto. Sus medidas en píxeles son de 450x375. Esta puede verse en la Introducción, en la Figura 1.1.

8.1. Cluster Intel Xeon E5-2630Lv2

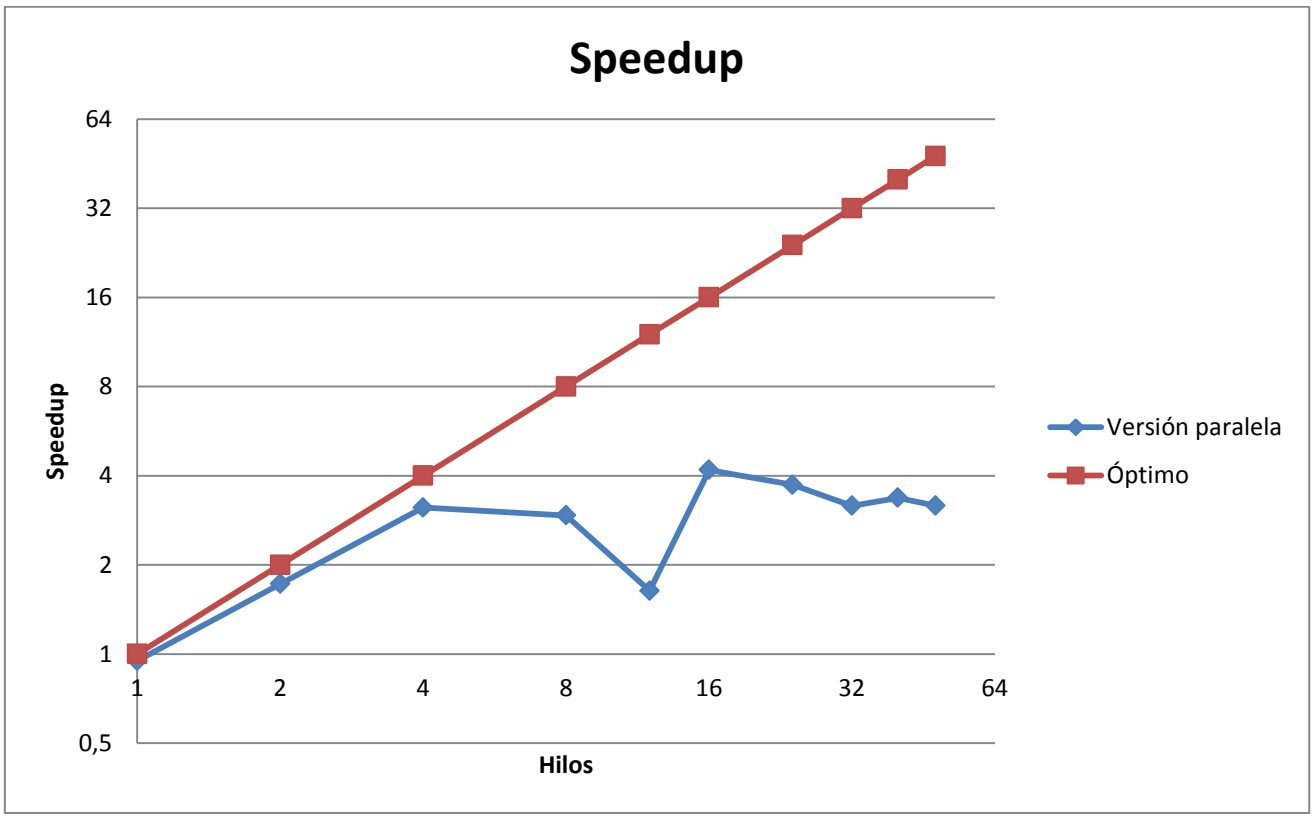
Para esta arquitectura, las pruebas realizadas incluyen la ejecución secuencial de todos los algoritmos de Stereo Matching estudiados en este proyecto, así como las de sus diferentes versiones paralelas. De cara a estas últimas, se ha tenido en cuenta el número de núcleos de los que dispone cada uno de los dos procesadores que conforman el servidor, es decir, en total habría disponibles 12 núcleos físicos. Por ello, las medidas para las versiones paralelas se han realizado sobre ejecuciones con 1, 2, 4, 8, 12, 16, 24, 32, 40 y 48 hilos, de modo que se puede estudiar cómo escalan las diferentes versiones y cómo se degrada el rendimiento a partir de la cantidad máxima de hilos que se pueden correr de forma simultánea.

Para cada una de las pruebas propuestas, se han realizado cinco mediciones para evitar la distorsión de los resultados por potenciales extremos atípicos, tomando la media para la elaboración de las representaciones gráficas.

8.1.1. Pixel-wise Matching

En este subapartado se discuten los resultados obtenidos para las ejecuciones de este algoritmo. A continuación se muestran en las Gráficas 8.1 y 8.2 sus tiempos de ejecución secuencial y paralela con los números de hilos mencionados, así como el *speedup* de las mediciones sobre el código optimizado. El *speedup* se define como el cociente del tiempo de cómputo de la versión secuencial y el de la versión paralela.

GRÁFICA 8.1: TIEMPOS DE EJECUCIÓN PARA PIXEL-WISE MATCHING



GRÁFICA 8.2: SPEEDUP PARA PIXEL-WISE MATCHING

Como podemos ver, en ambas gráficas el número de hilos se representa en escala logarítmica en base dos, ya que para las medidas realizadas para cantidades

elevadas se han elegido números más dispersos entre sí. También se muestra en la misma escala el eje del factor de *speedup*, ya que este debería mantener cierta proporcionalidad con el número de hilos utilizados.

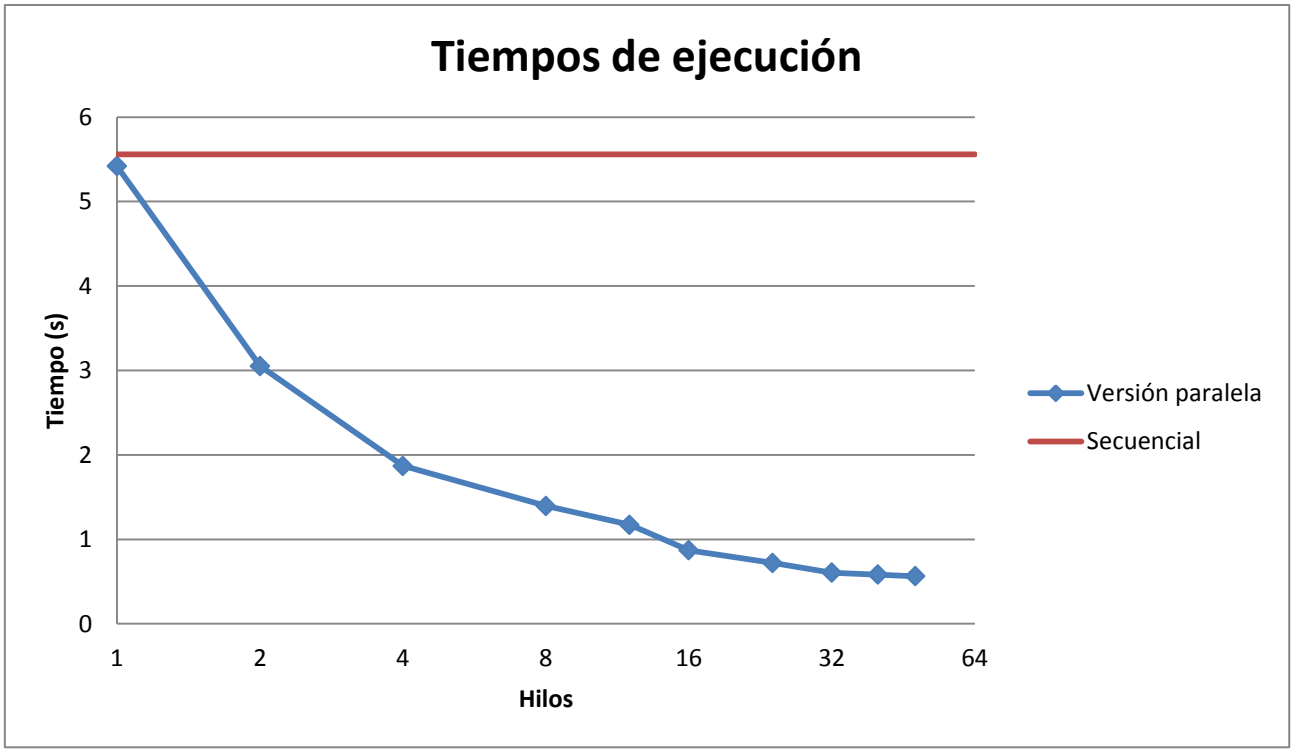
Para la primera gráfica se ha añadido, además, una línea horizontal que indica el rendimiento obtenido con la versión secuencial del algoritmo, para tomar como referencia al ver el resto de medidas. Por otra parte, en la segunda se ha incluido una recta que representa el factor de *speedup* ideal, que sería igual al número de hilos utilizados y, aunque resulte utópico, es el valor que se debería aspirar a coincidir y se puede tomar como referencia para saber cuánto rendimiento se ha perdido en la paralelización.

Dicho esto, se puede ver que el algoritmo escala muy bien (de acorde a lo esperado, ya que el reparto de carga de trabajo entre los hilos resulta trivial y es equitativo) para cantidades de hilos menores que el número de núcleos de cada procesador: 6. A partir de ese punto se sufre una disminución de la ganancia que podría tener que ver con la necesidad de uso del segundo procesador del cluster y con la comunicación entre ambos que esto implica. Además, con una carga de trabajo tan reducida, el coste de arrancar los hilos a partir de cierto punto podría no verse compensado por la ganancia que aporta el reparto de los cálculos.

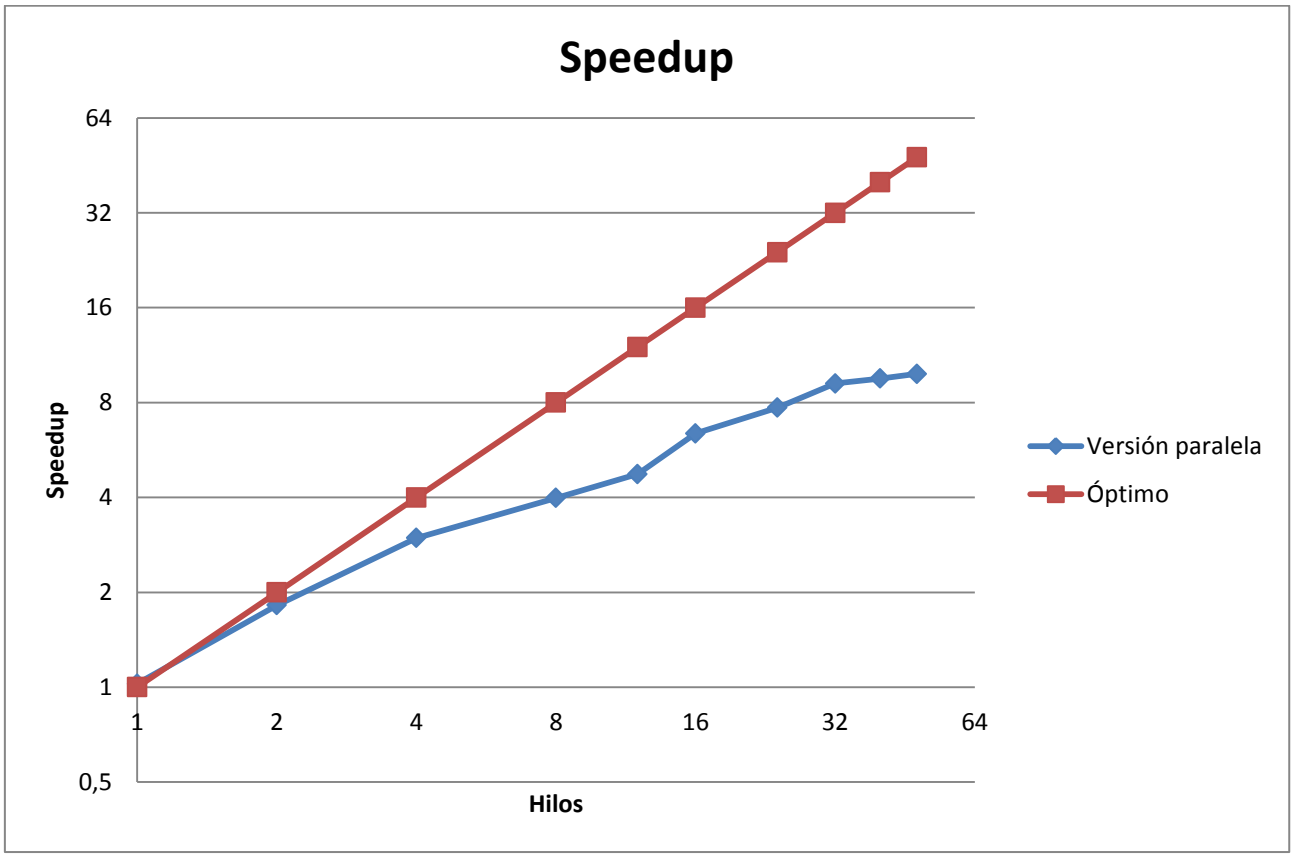
También podemos ver una clara tendencia descendente a partir del punto en el que todos los núcleos lógicos del cluster de procesadores están cubiertos, debida a la degradación que sufre el rendimiento con una planificación del sistema para la ejecución de los hilos que cambia de contexto más de lo que debería.

8.1.2. Patch-wise Matching

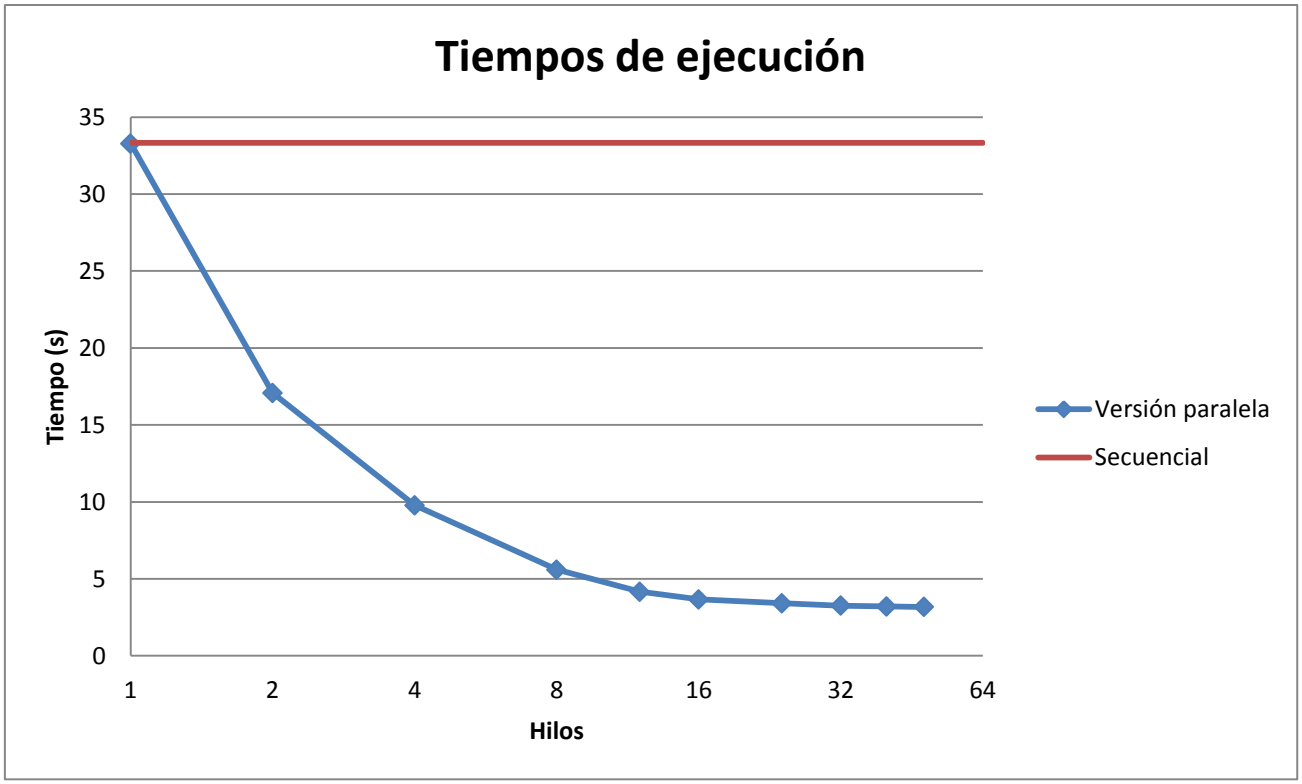
En este sub-apartado se discuten los resultados obtenidos de la ejecución de este algoritmo en ambas variantes, ya que los comportamientos son similares entre sí y es posible analizarlos, por lo tanto, de forma conjunta. Se seguirá la misma estructura que en el apartado anterior, mostrando a continuación las mismas representaciones para estos datos, antes de su análisis:



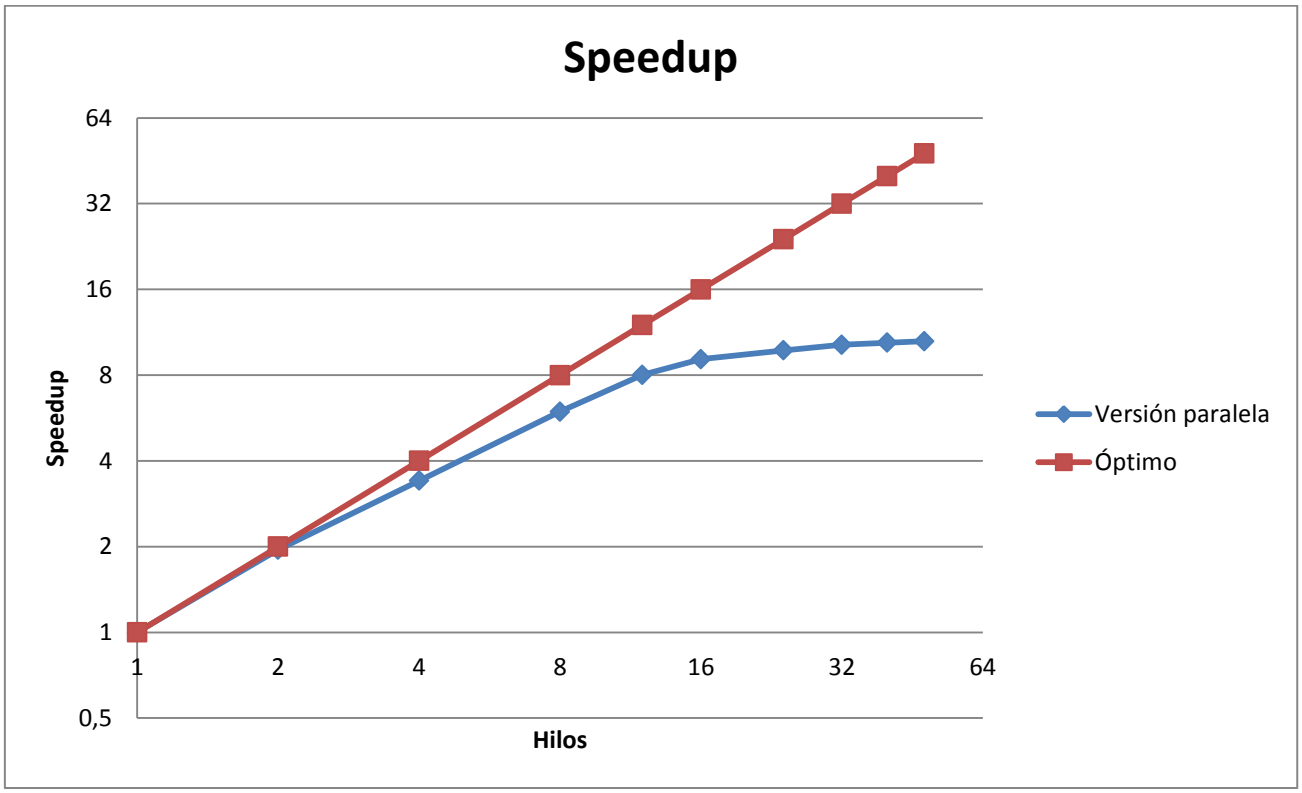
GRÁFICA 8.3: TIEMPOS DE EJECUCIÓN PARA *SUM OF ABSOLUTE DIFFERENCES*, XEON



GRÁFICA 8.4: SPEEDUP PARA *SUM OF ABSOLUTE DIFFERENCES*, XEON



GRÁFICA 8.5: TIEMPOS DE EJECUCIÓN PARA *ADAPTIVE SUPPORT-WEIGHT*, XEON



GRÁFICA 8.6: SPEEDUP PARA *ADAPTIVE SUPPORT-WEIGHT*, XEON

Como se puede observar, la escalada de rendimiento en estos casos es casi lineal, acercándose mucho al óptimo. De hecho, se mantiene un aumento de la

ganancia incluso más allá del límite razonable, que es la ocupación de todos los núcleos lógicos. Esto podría deberse a que, con un mayor nivel de reparto, las franjas asignadas a cada hilo son más estrechas, con lo que las probabilidades de que uno cargue en memoria caché un dato que el otro necesita aumentan. Esto es posible debido a que, para la consideración de un candidato, se toma un parche a su alrededor que podría abarcar píxeles dentro de otra franja. Con franjas más estrechas, la proporción de elementos comunes entre ellas respecto a los que no lo son será mayor.

Por otra parte, podemos ver que para cantidades de hilos próximas al número de núcleos físicos de cada procesador, la ganancia se degrada un poco más para el primer método. Esto podría tener que ver con el hecho que se ha comentado ya en el subapartado anterior respecto al uso del segundo procesador del cluster, y tienen que organizarse para el reparto del trabajo y la puesta en común de resultados, lo cual no tendría un efecto tan importante proporcionalmente sobre el segundo método debido a que el tiempo consumido para los cálculos por este es mucho mayor, del orden de cinco veces más para las versiones secuenciales de ambos.

En cualquier caso, podemos ver que para los dos métodos se llega a obtener un factor de *speedup* de alrededor de 10 en el servidor de 12 núcleos compuesto por dos procesadores diferentes, lo cual se puede considerar como un valor razonablemente elevado, acorde a la hipótesis realizada al respecto en el capítulo en el que se describían los algoritmos considerados.

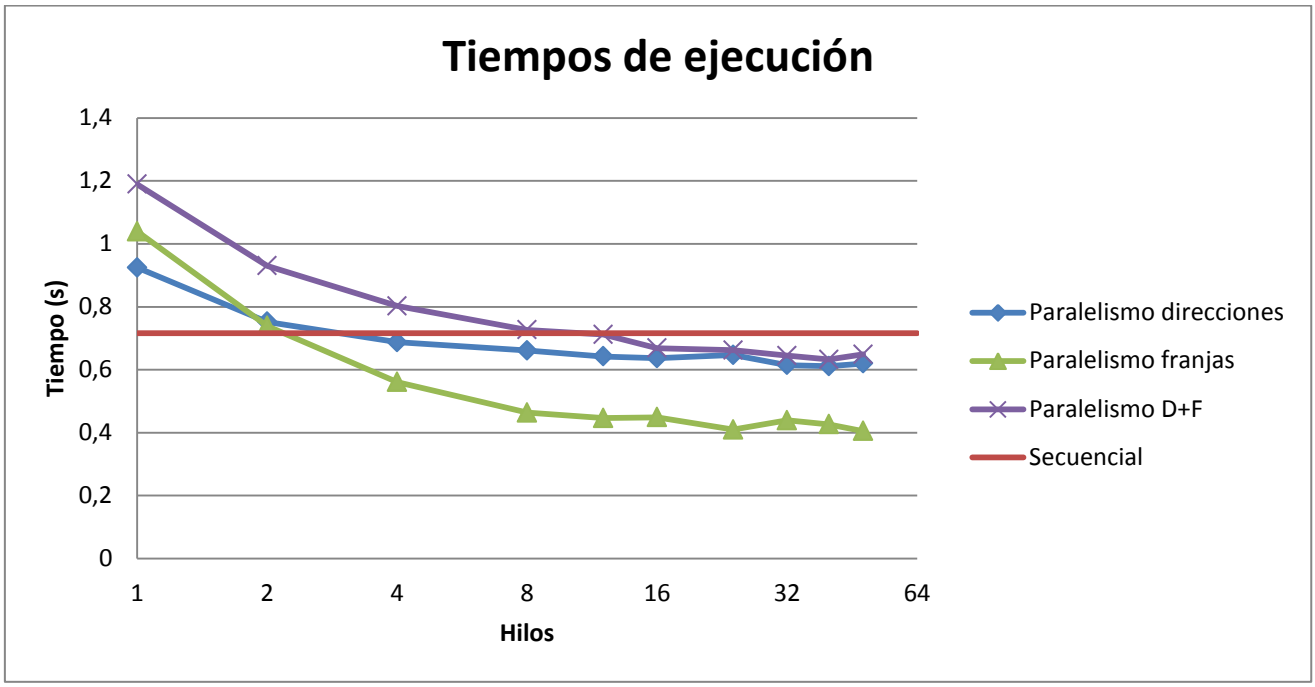
8.1.3. Semi-Global Matching

En este subapartado se analiza el algoritmo más interesante, en lo que a estrategias de paralelización se refiere, de entre los considerados. Como ya se ha explicado, su paralelización resulta significativamente más compleja que la de los otros dos algoritmos. Esto lleva a la separación en los tres bloques mencionados en la Sección 7.2, para los cuales se han realizado medidas por separado, en adición a la del total de tiempo consumido.

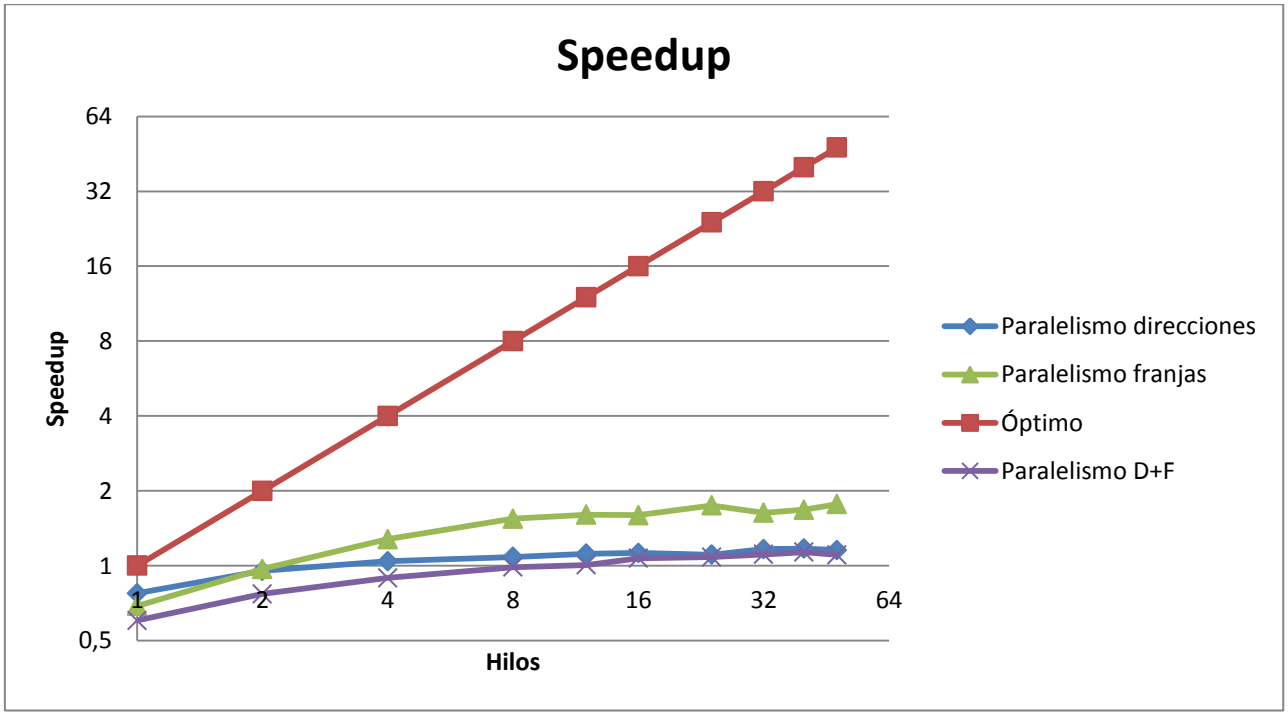
Por los motivos anteriores, este subapartado no seguirá la misma estructuración que los que le preceden. Se comenzará por el mismo análisis que el realizado para los otros algoritmos, pero además, a continuación se añadirán las gráficas de rendimiento para cada uno de los tres bloques definidos y se comentarán los resultados correspondientes. Además, para cada uno de los bloques existen tres estrategias de paralelización diferentes, de modo que estas también serán comentadas y comparadas entre sí.

También se añadirá un breve análisis sobre la posibilidad de cambiar el reparto de los hilos en la tercera estrategia de paralelización, variando las cantidades de ellos que se asignan al bucle externo y al interno.

A continuación se muestran en las Gráficas 8.7 y 8.8 los tiempos de ejecución y *speedup* obtenidos para el total de la ejecución:



GRÁFICA 8.7: TIEMPOS DE EJECUCIÓN PARA SEMI-GLOBAL MATCHING (TOTAL), XEON



GRÁFICA 8.8: SPEEDUP PARA SEMI-GLOBAL MATCHING (TOTAL), XEON

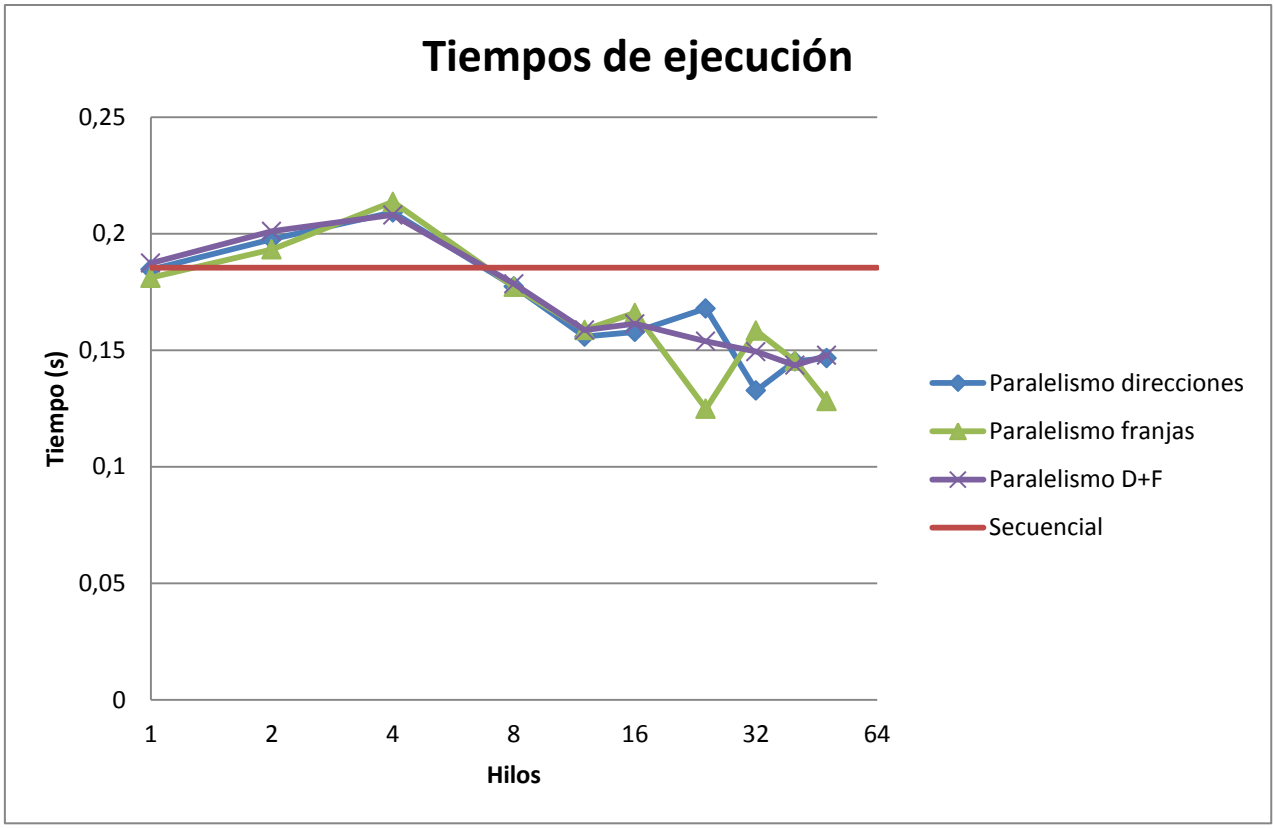
Como se puede observar, entre los resultados obtenidos de las diferentes estrategias de paralelización hay un patrón común: todas ofrecen un rendimiento sensiblemente peor que la versión secuencial para ejecuciones con un hilo y,

ligeramente también para aquéllas con dos hilos. Esto es algo que cabía esperar, como se ha comentado ya en el capítulo anterior, debido a que todas incluyen *overheads* de cálculo, y el tiempo total de ejecución es muy pequeño, lo que convierte a estos en más significativos. Para la paralelización por franjas, los cálculos añadidos provienen del reparto inicial de las diagonales y las franjas horizontales y verticales. Para la estrategia basada únicamente en la simultaneidad de cálculo de las soluciones parciales, el origen es la reducción basada en *mutex* necesaria para calcular la solución agregada sin carreras críticas. Para el último método, existen ambos tipos de *overhead*, lo que lo posiciona desfavorablemente para ejecuciones con pocos hilos.

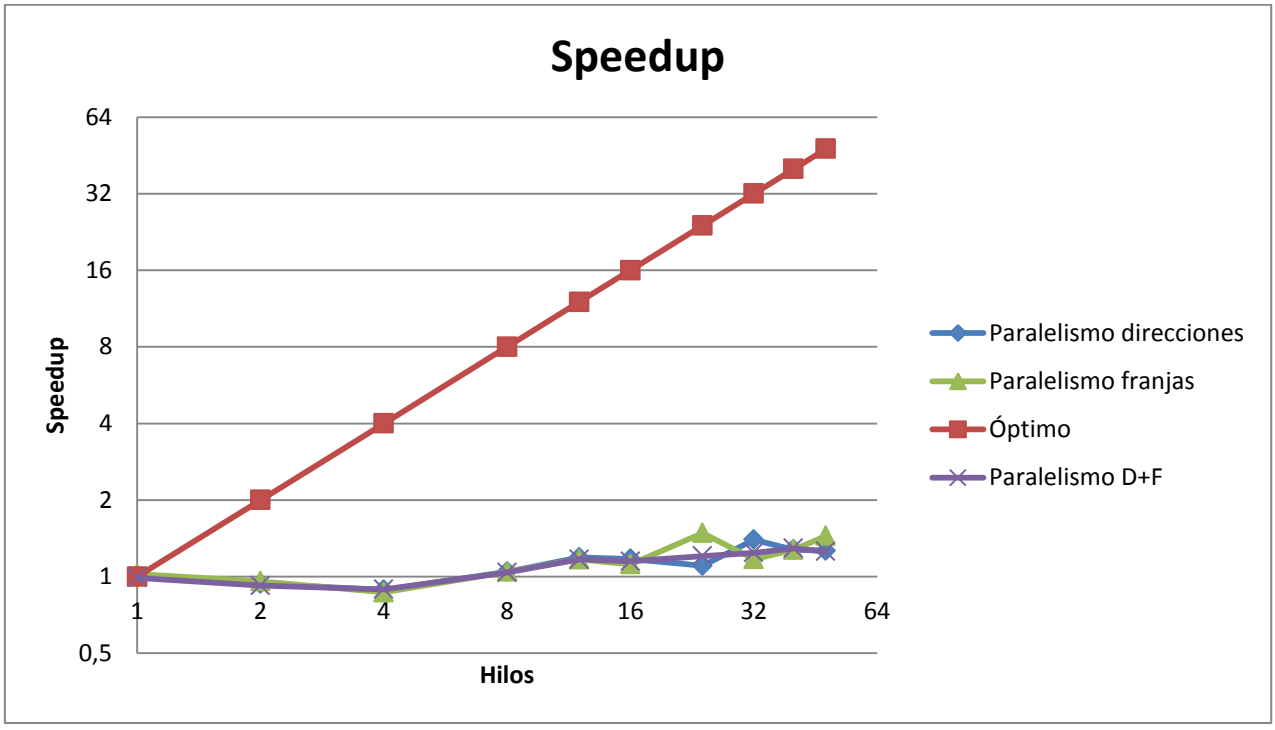
Además de lo ya comentado, se puede observar que, en general para las tres estrategias, la pendiente de ganancia respecto al número de hilos utilizados es claramente inferior a los resultados ideales, y que se degrada considerablemente al superar la cantidad de hilos el número de núcleos físicos de un procesador. Este fenómeno afecta, sobre todo, a la paralelización por direcciones, ya que el núcleo de los cálculos no aumenta su rendimiento en proporción al número de *threads* utilizados a partir de 8 (no hay más soluciones parciales que requieran ser calculadas de forma simultánea). Los otros bloques sí tendrían ligeras mejorías, pero el aporte de estos al total del tiempo de ejecución es limitado, con lo que este método, que comienza para un único hilo como el mejor de los tres, es alcanzado en las últimas ejecuciones con muchos hilos por la paralelización combinada, que comienza como el peor.

Se puede ver, por lo tanto, que para una ejecución por defecto en esta arquitectura, la mejor alternativa es utilizar el sistema de franjas.

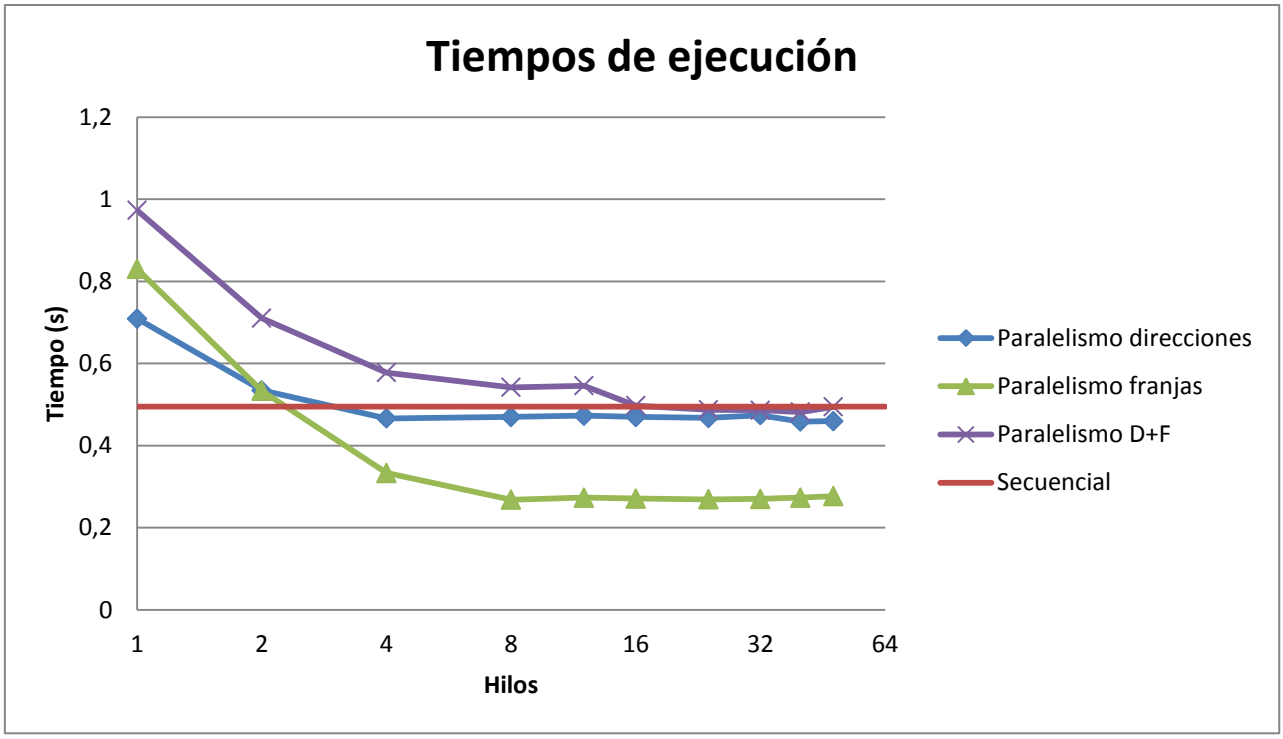
A continuación se desglosan estos tiempos en los tres bloques ya mencionados. Esto se ve ilustrado en las Gráficas 8.9, 8.10, 8.11, 8.12, 8.13 y 8.14.



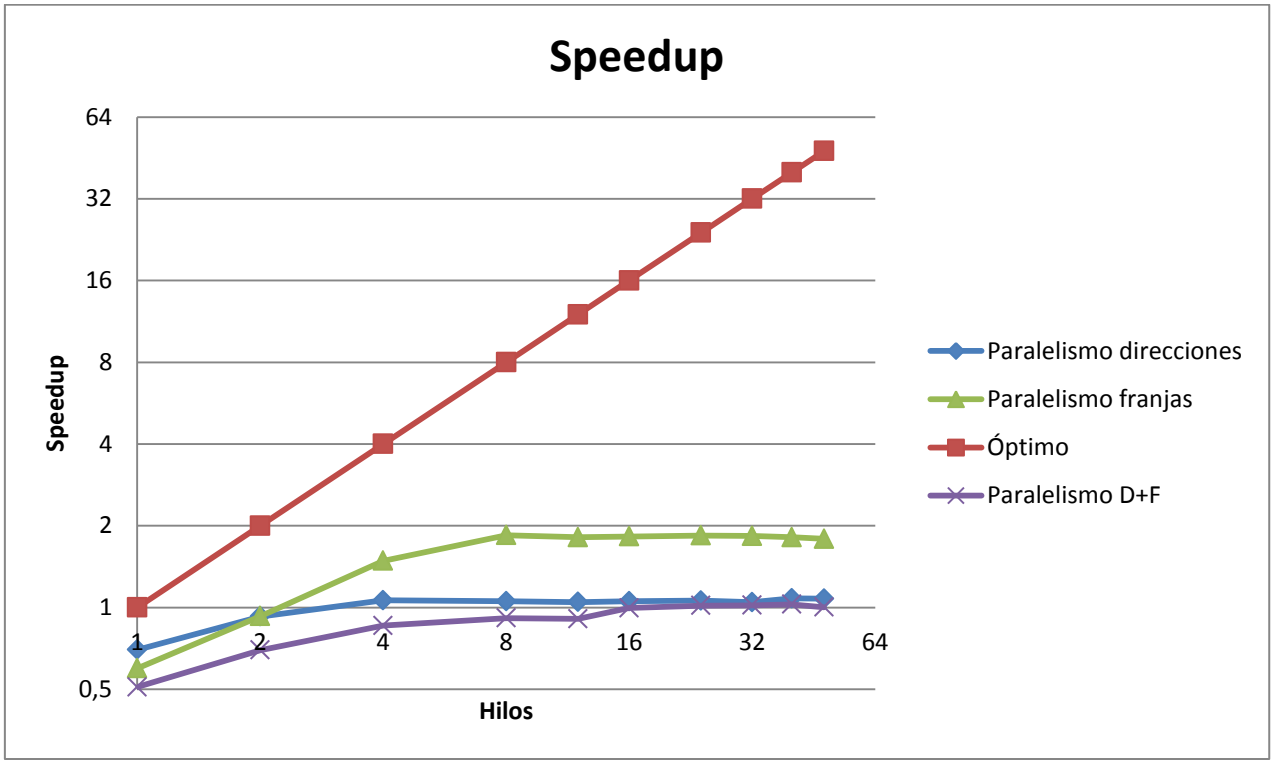
GRÁFICA 8.9: TIEMPOS DE EJECUCIÓN PARA SEMI-GLOBAL MATCHING (BLOQUE 1), XEON



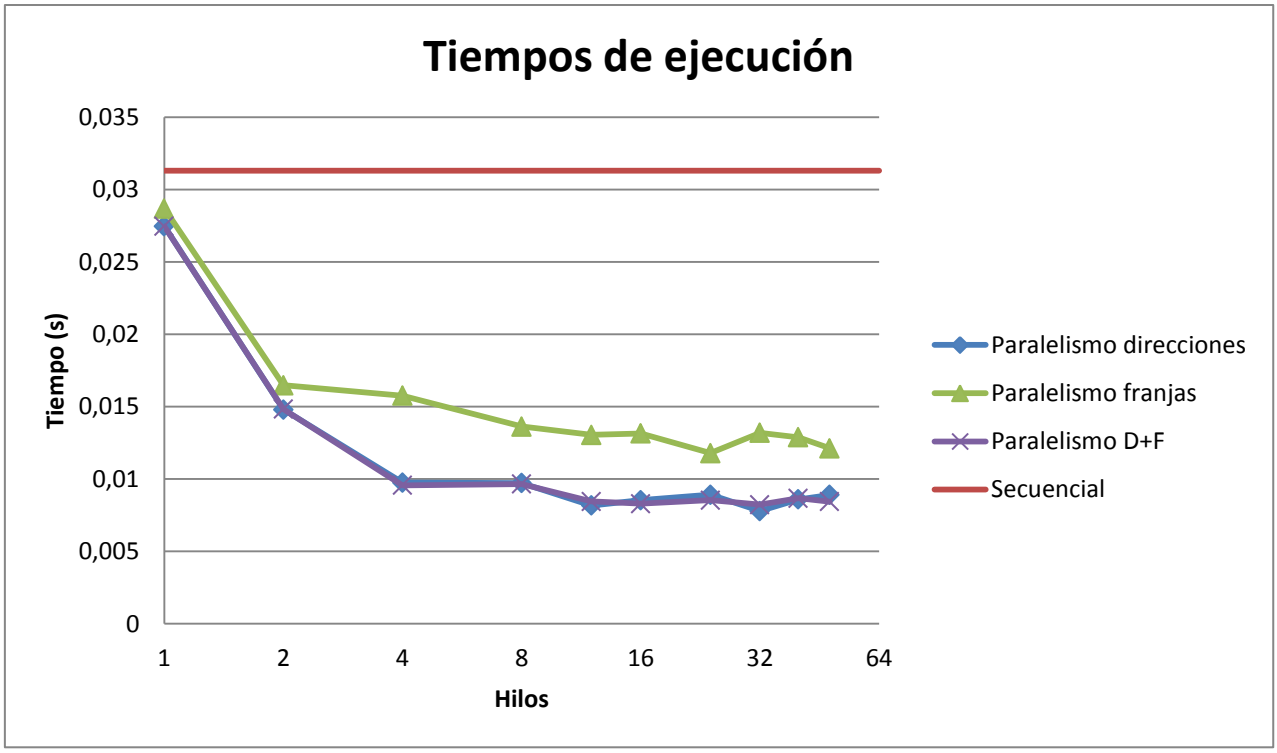
GRÁFICA 8.10: SPEEDUP PARA SEMI-GLOBAL MATCHING (BLOQUE 1), XEON



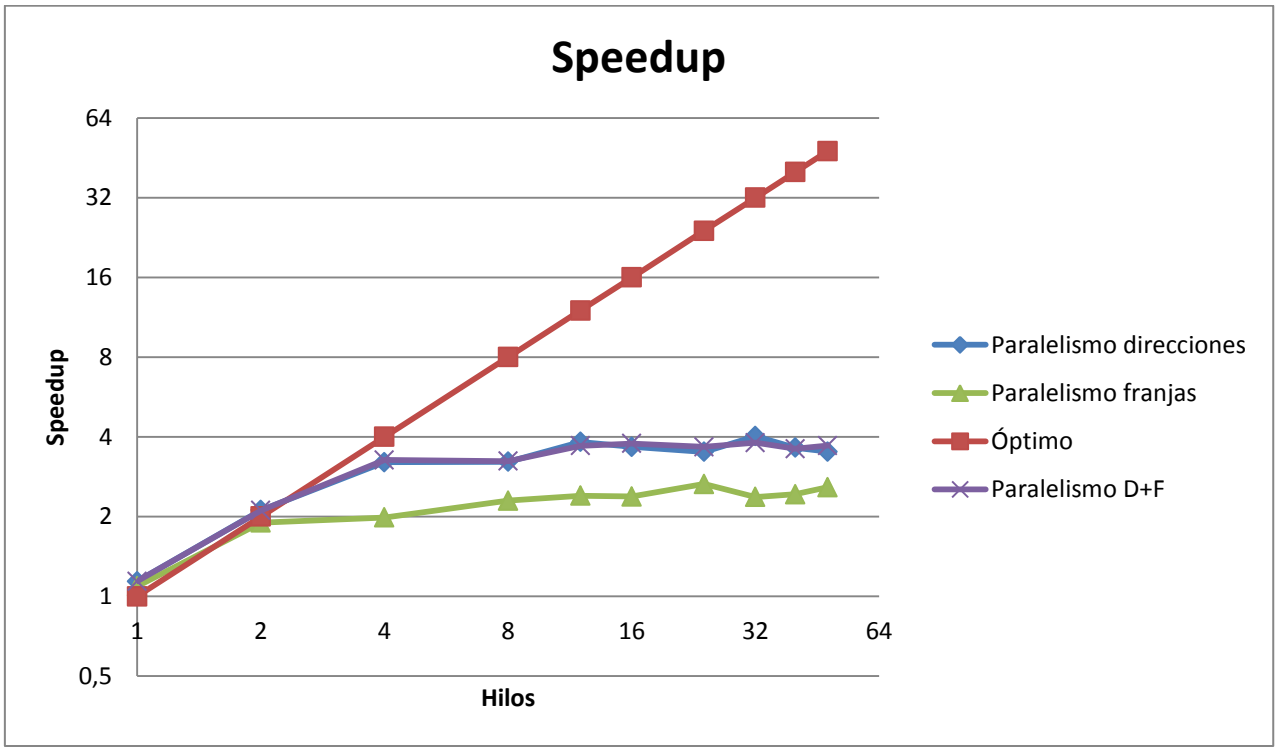
GRÁFICA 8.11: TIEMPOS DE EJECUCIÓN PARA SEMI-GLOBAL MATCHING (BLOQUE 2), XEON



GRÁFICA 8.12: SPEEDUP PARA SEMI-GLOBAL MATCHING (BLOQUE 2), XEON



GRÁFICA 8.13: TIEMPOS DE EJECUCIÓN PARA SEMI-GLOBAL MATCHING (BLOQUE 3), XEON



GRÁFICA 8.14: SPEEDUP PARA SEMI-GLOBAL MATCHING (BLOQUE 3), XEON

Respecto al primer bloque (Figuras 8.9 y 8.10), podemos observar que los rendimientos de las diferentes versiones escalan parejos entre sí, debido a que la paralelización de esta parte es idéntica para todos. Se observa una ligera degradación del rendimiento para cantidades de hilos bajas, que puede deberse al esfuerzo que

supone crear los hilos y que no se ve compensado por el ahorro en una zona donde los cálculos realizados son tan poco costosos.

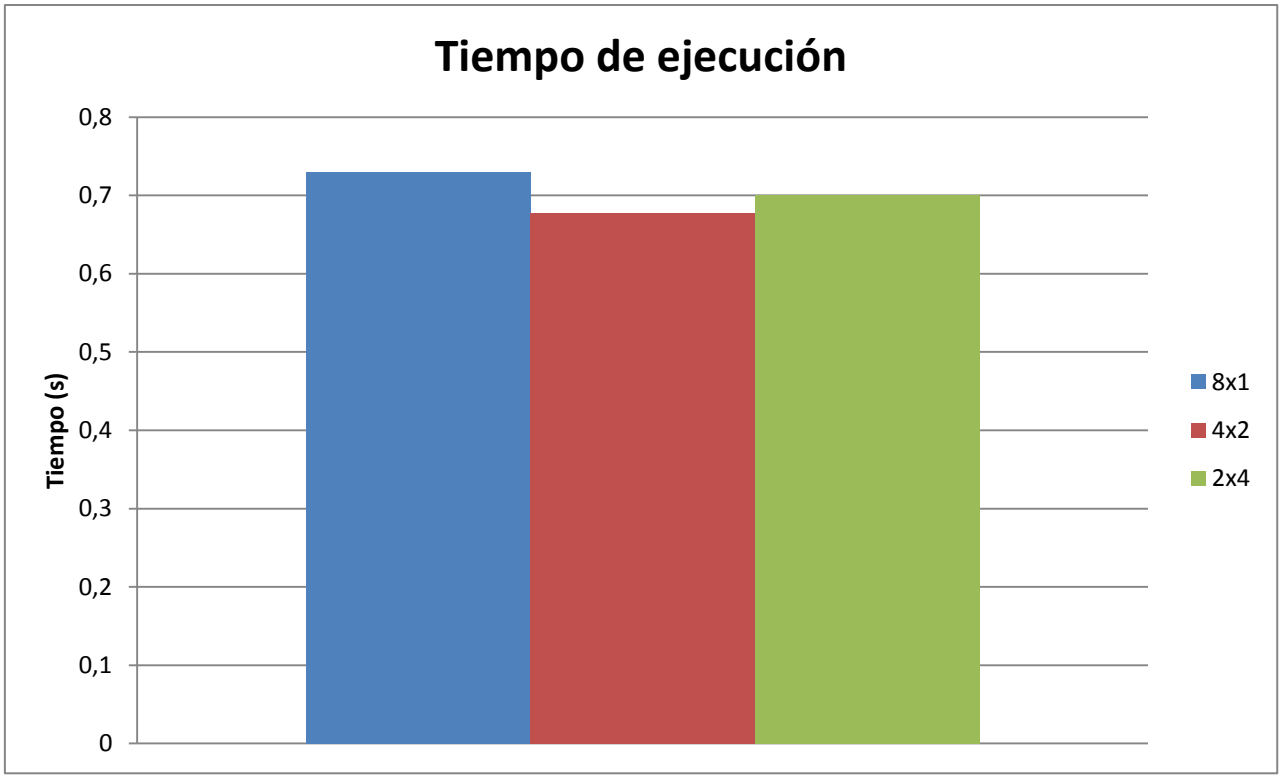
En el tercero (Figuras 8.13 y 8.14) debería suceder algo análogo, ya que la paralelización tiene las mismas características. No obstante, ya se ha realizado la ejecución de todo el resto del programa con anterioridad, de modo que se presentan algunas diferencias.

Primeramente, no se aprecia el despunte de tiempo consumido para cantidades bajas de hilos que se ve en el primer bloque, probablemente debido a que los hilos ya han sido creados con anterioridad y se aprovecha para lanzarlos con menos esfuerzo en esta ocasión, con lo que el *overhead* correspondiente es mínimo. Unido a esto, se espera que haya datos disponibles en las memorias caché desde el principio.

Por otra parte, se puede observar que la ejecución de la estrategia basada en el sistema de franjas arroja peores resultados que las otras dos. Esto podría deberse a que en estas últimas la reducción de la solución definitiva se realiza al final y no durante todo el proceso, con lo que tiene más probabilidades de encontrarse en caché todavía al comenzar la ejecución de este bloque o, al menos, de que gran parte de sus elementos lo estén. Además, puede afectar el posible desbalanceo de la carga propio de esta versión.

Por último, el segundo bloque (Figuras 8.11 y 8.12) es decisivo para el tiempo de ejecución total, ya que es el que contiene el núcleo de los cálculos del algoritmo y, por lo tanto, el que más tiempo consume. Podemos ver que las tendencias de las gráficas correspondientes son análogas a las que provienen de la ejecución conjunta de los tres bloques. Dichas tendencias se explican, por lo tanto, con los mismos argumentos ofrecidos en el comentario de las dos primeras gráficas de este subapartado.

Como conclusión de este apartado, podemos ver los resultados de un pequeño experimento secundario que consiste en la modificación del reparto de los hilos entre el lazo externo y el interno en la paralelización combinada. Se ha decidido realizar la comparación para la ejecución con 8 hilos. Utilizamos primero la distribución habitual, que toma todos los hilos posibles hasta 8 para el lazo externo y luego reparte equitativamente los restantes para cada una de las soluciones parciales, tomando por lo tanto, en este caso, los ocho para el exterior y 1 para cada solución parcial (se computarán de forma secuencial). Luego se ha probado con las variantes en las que el lazo externo utiliza 4 hilos y el interno 2, y viceversa. Los resultados de rendimiento son los que se ven en el siguiente gráfico de barras:



GRÁFICA 8.15: COMPARATIVA DE DIFERENTES REPARTOS ENTRE LAZOS EXTERNO E INTERNO, XEON

Podemos observar que existen pequeñas diferencias de rendimiento, pero que en proporción al tiempo de ejecución son mínimas. No parece, por lo tanto, que la selección de una estrategia u otra vaya a suponer diferencias notorias en la ejecución del algoritmo.

8.2. Intel Xeon Phi 7120P

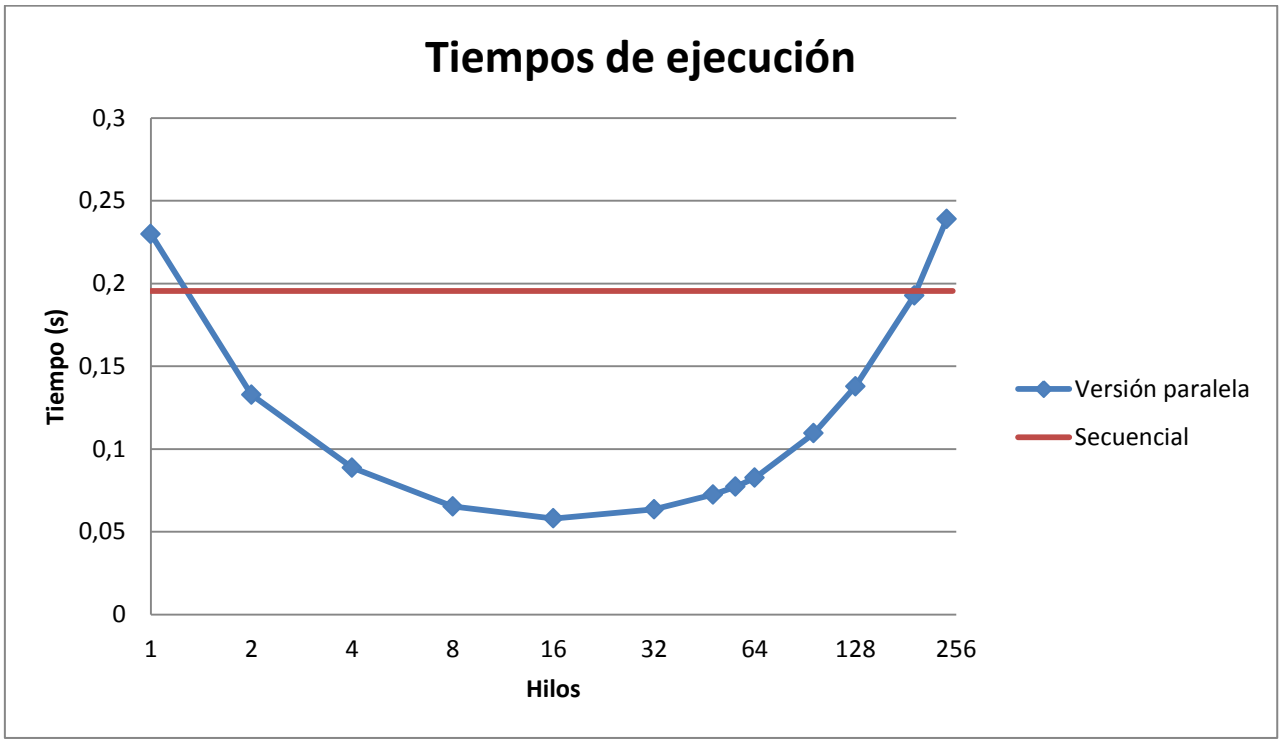
Para esta arquitectura se han realizado unas pruebas análogas a las descritas anteriormente, teniendo en cuenta que ahora el número de núcleos es mucho mayor: 61. También cambia la cantidad de hilos que puede manejar cada núcleo físico por medio de la tecnología Hyper-Threading, que ascienden a 4. Por ello, se ha decidido realizar las mediciones para ejecuciones con 1, 2, 4, 8, 16, 32, 48, 56, 64, 96, 128, 192 y 240 hilos.

Por otra parte, también se añadirá respecto al apartado anterior una breve sección en la que se compararán los diferentes modos de afinidad disponibles para este procesador.

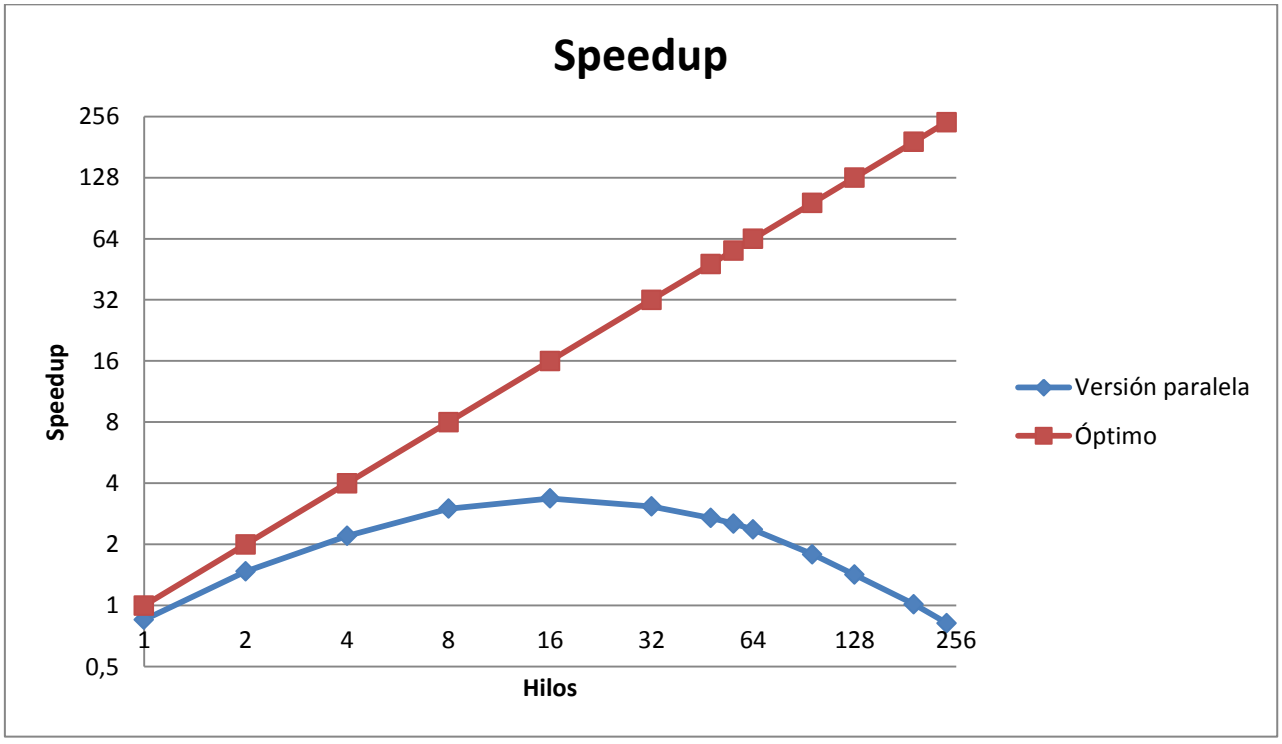
Como recordatorio previo al análisis de resultados, hay que tener en cuenta respecto a la arquitectura anterior que los núcleos de este sistema ofrecen un rendimiento bastante limitado, y que la ganancia respecto al uso de un procesador se obtiene para ejecuciones altamente paralelas. Se espera, por lo tanto, que una ejecución con pocos hilos desemboque en resultados peores que en el cluster anterior.

8.2.1. Pixel-wise Matching

Siguiendo el esquema propuesto en el anterior apartado, a continuación se muestran en las Gráficas 8.16 y 8.17 los resultados de la ejecución de las versiones secuencial y paralela de este algoritmo.



GRÁFICA 8.16: TIEMPOS DE EJECUCIÓN PARA PIXEL-WISE MATCHING, PHI

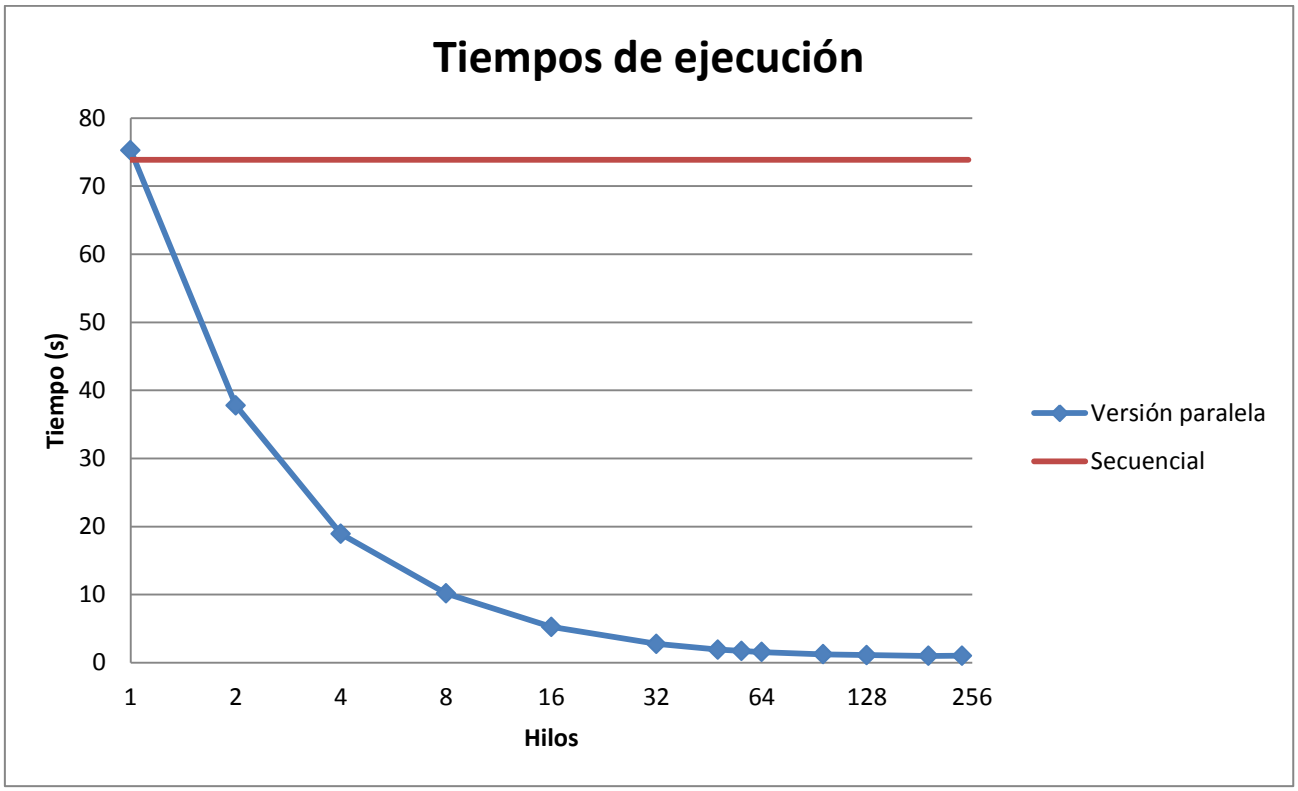


GRÁFICA 8.17: SPEEDUP PARA PIXEL-WISE MATCHING, PHI

Podemos observar que el aumento de rendimiento es casi lineal al principio, pero que se degrada muy rápidamente hasta volver a ofrecer resultados peores que la versión secuencial. Esto se puede deber, como ya hemos comentado con anterioridad, a que para una ejecución tan breve pueda no compensar el esfuerzo de crear tantos hilos, ya que podría consumir más tiempo que el que ahorran con una distribución de carga un poco mejor. Este algoritmo parece, por lo tanto, más apropiado para su ejecución con una cantidad moderada de hilos o para el procesado de imágenes de mayor tamaño.

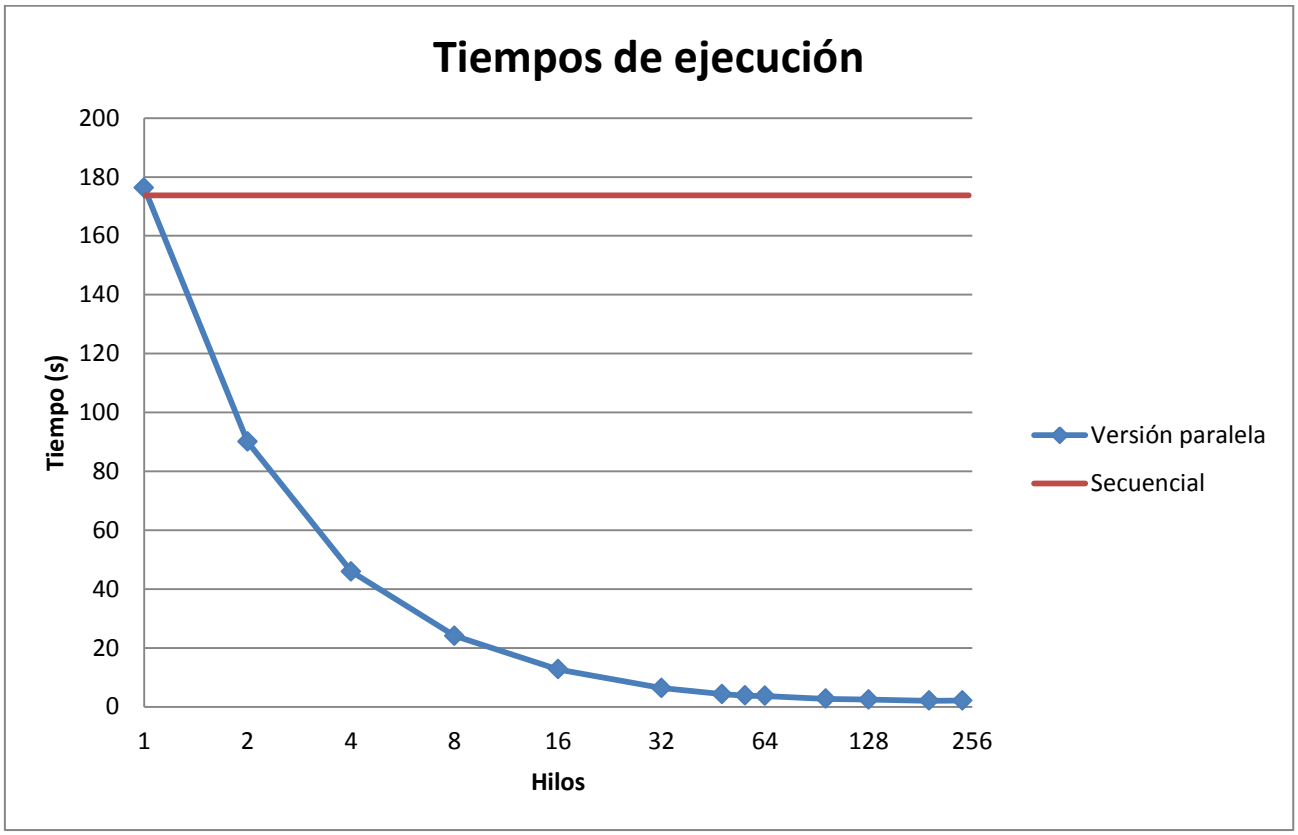
8.2.2. Patch-wise Matching

Una vez más, agrupamos bajo el mismo subapartado ambas variaciones de este algoritmo, ya que sus resultados siguen tendencias muy similares entre sí, como se puede ver en las Gráficas 8.18, 8.19, 8.20 y 8.21.

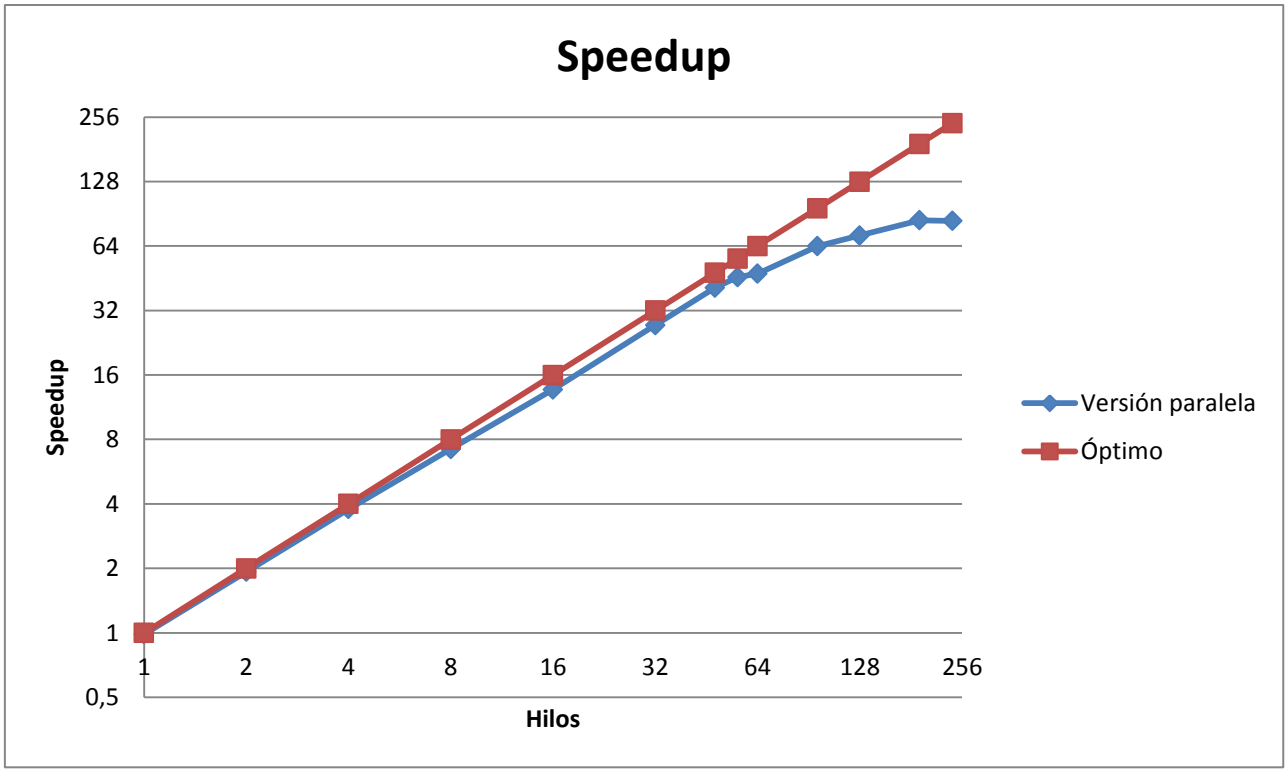


GRÁFICA 8.18: TIEMPOS DE EJECUCIÓN PARA *SUM OF ABSOLUTE DIFFERENCES*, PHI

GRÁFICA 8.19: SPEEDUP PARA *SUM OF ABSOLUTE DIFFERENCES*, PHI



GRÁFICA 8.20: TIEMPOS DE EJECUCIÓN PARA *ADAPTIVE SUPPORT-WEIGHT*, PHI



GRÁFICA 8.21: SPEEDUP PARA *ADAPTIVE SUPPORT-WEIGHT*, PHI

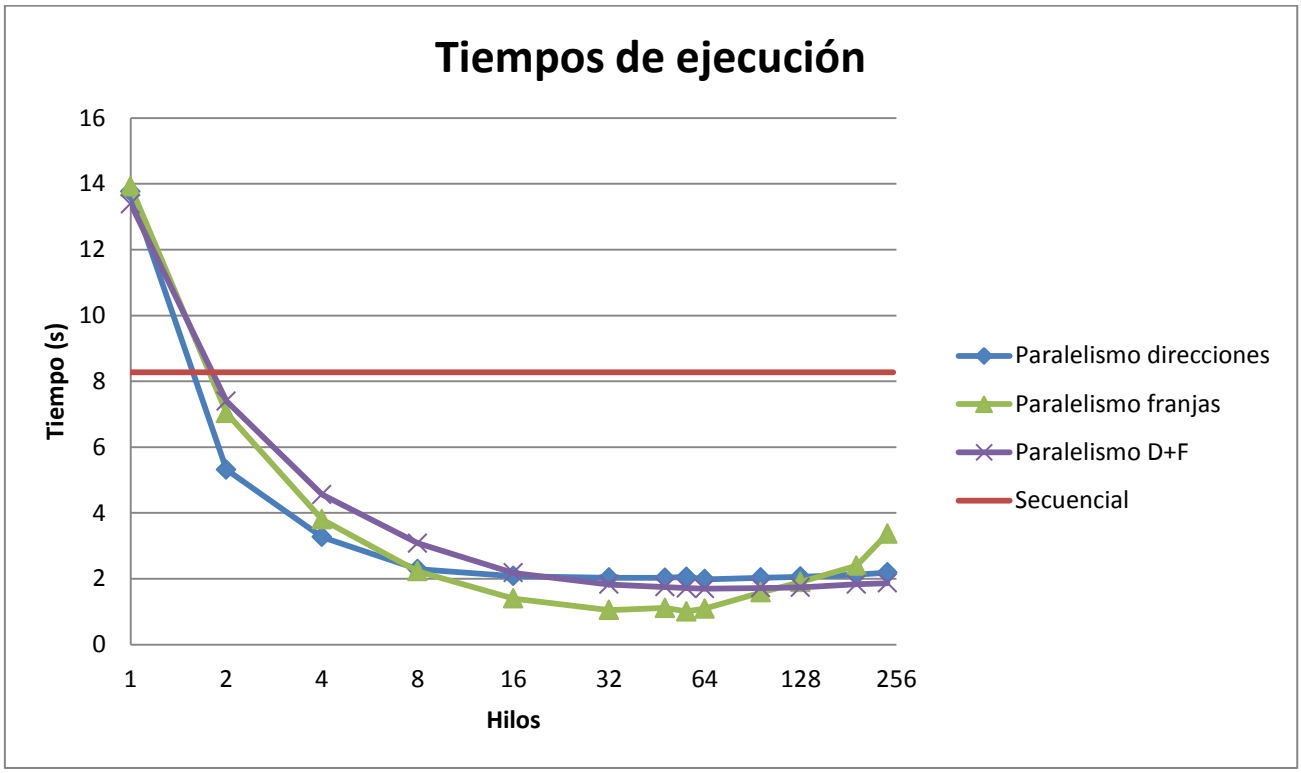
Como cabía esperar, podemos ver que el *speedup* obtenido en ambas versiones es cercano al óptimo hasta el uso de cantidades muy elevadas de hilos, alrededor de 100, punto en el que se degrada un poco la pendiente de la ganancia respecto al número de *threads* utilizados, que sigue, no obstante, siendo bastante elevada. Esta ligera suavización de la ganancia es probable que tenga su causa en el hecho de que se ha alcanzado el punto de máximo aprovechamiento de la tecnología Hyper-Threading, algo antes de los cuatro hilos que teóricamente permite correr en cada núcleo de forma simultánea.

Esto se ve favorecido por el hecho de que, en este procesador, como ya hemos dicho, los núcleos tienen prestaciones bastante limitadas individualmente, por lo que el tiempo de ejecución secuencial es considerablemente elevado, con lo que el esfuerzo de crear los hilos es razonablemente reducido en proporción. De este modo, se aprovecha casi por completo el reparto poco costoso y equitativo que se realiza de la imagen.

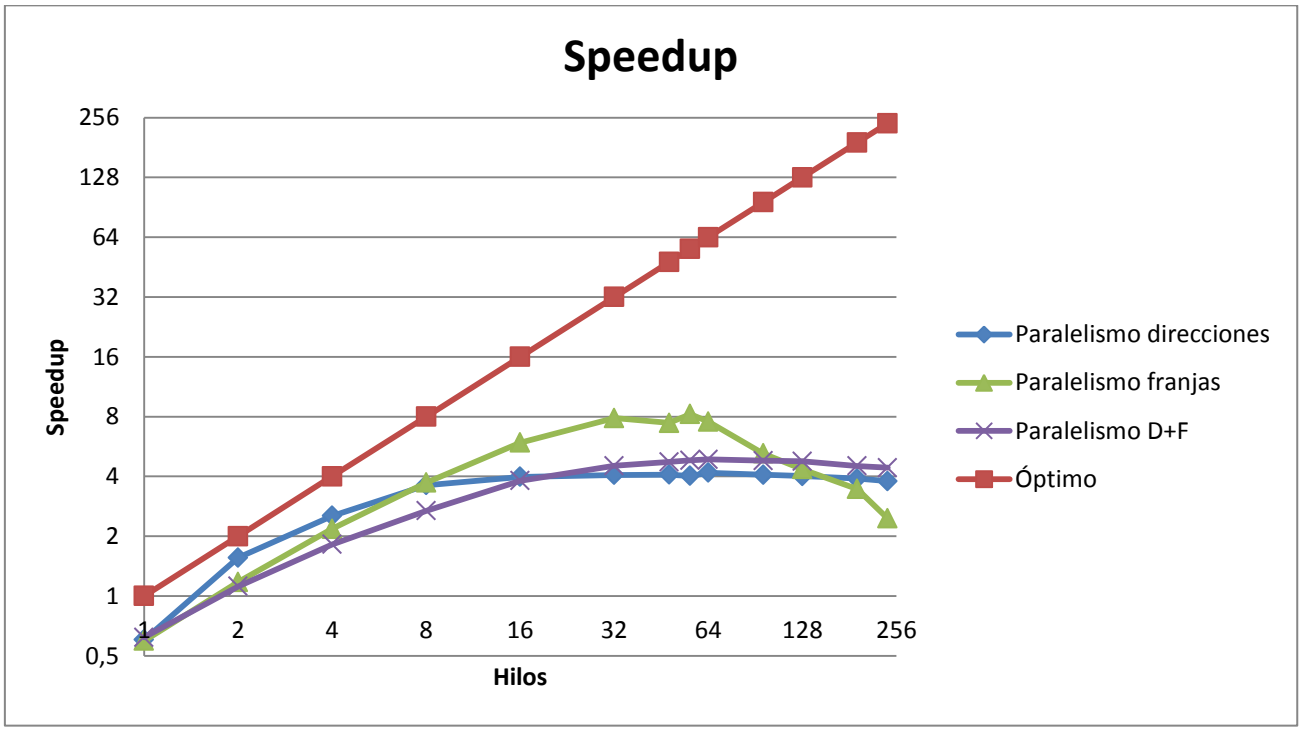
Cabe destacar el extremo al que llega la ganancia ofrecida por el paralelismo, alcanzando un factor de *speedup* de más de 74 para el método SAD, y más de 84 para ASW, ambos realmente elevados.

8.2.3. Semi-Global Matching

Siguiendo la estructuración ya definida para los subapartados de esta sección, a continuación se muestran en las Gráficas 8.22 y 8.23 los resultados para la ejecución de las distintas versiones de este algoritmo.



GRÁFICA 8.22: TIEMPOS DE EJECUCIÓN PARA SEMI-GLOBAL MATCHING, PHI



GRÁFICA 8.23: SPEEDUP PARA SEMI-GLOBAL MATCHING, PHI

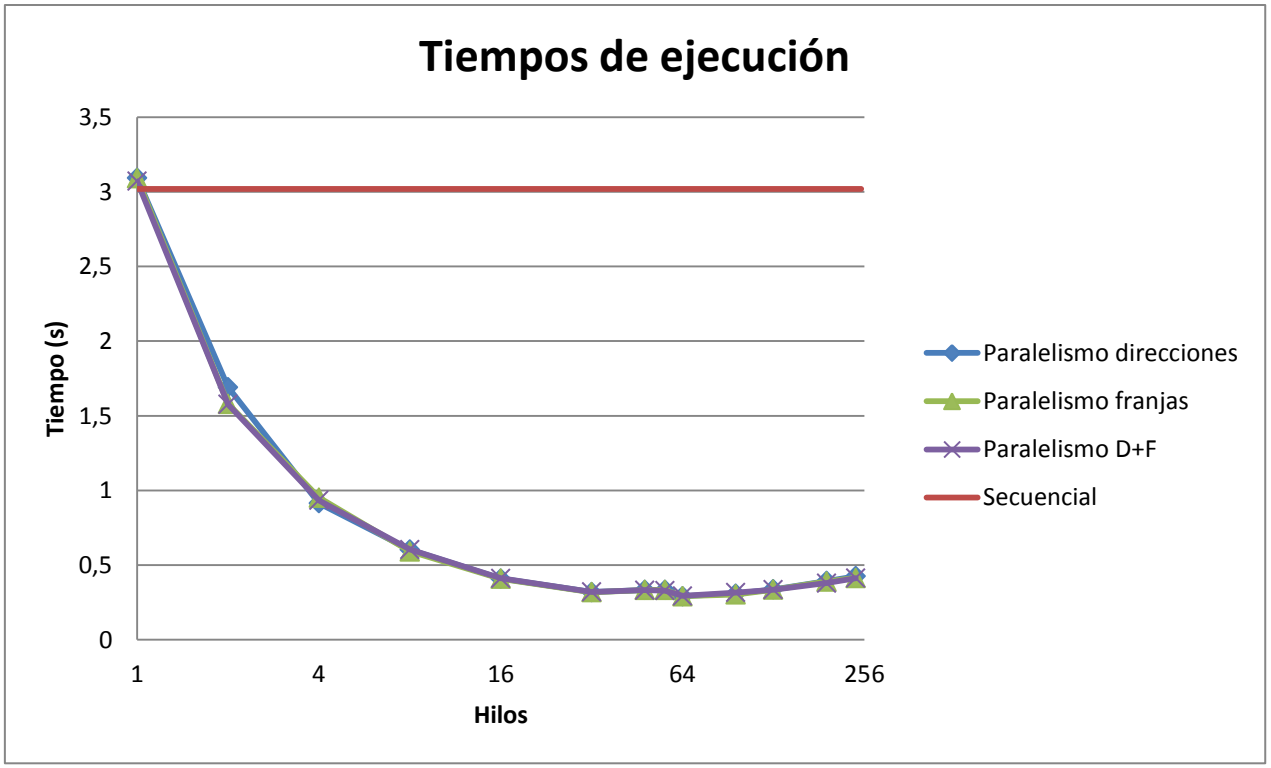
Podemos ver que, igual que el anterior, escalan bien para cantidades moderadas de hilos de ejecución pero, en este caso, el *speedup* se degrada antes. Las causas deben ser analizadas por separado para las tres aproximaciones utilizadas.

En la estrategia por franjas, la suavización y posterior caída de la ganancia se debe al desbalanceo que existe en el reparto de las diagonales, debido al hecho de que es necesario tomar diagonales enteras con lo que, para cantidades de hilos lo suficientemente grandes, las cargas de trabajo para algunos de ellos van a ser claramente inferiores a las de otros, con lo que no se mejora el rendimiento lanzando más *threads*, acción que supone, además, un esfuerzo extra para la máquina, de modo que los resultados empeoran.

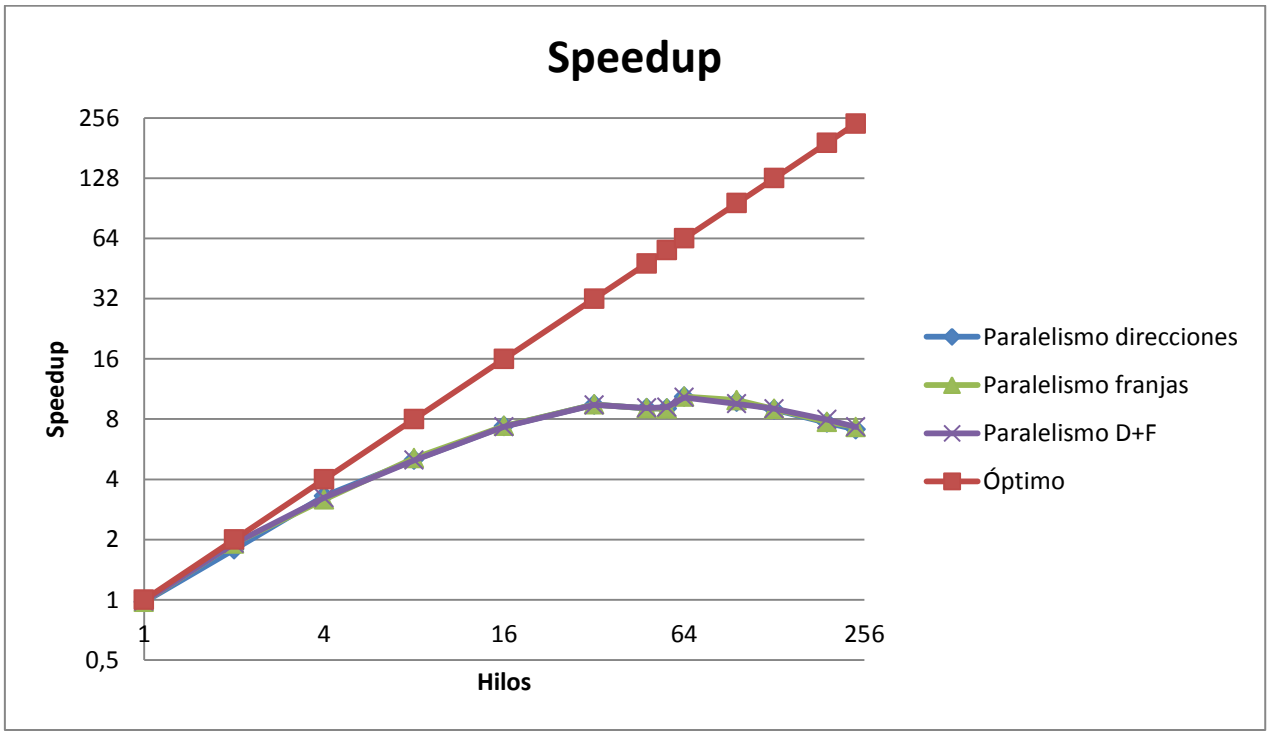
Para la paralelización por direcciones, observamos el mismo efecto que ya se podía apreciar en los resultados para la primera arquitectura considerada: a partir de los ocho hilos, uno por cada solución parcial, la única ganancia que se obtiene de lanzar más es en los bloques primero y tercero que, como ya hemos visto, no suponen una diferencia importante en el tiempo de ejecución total, que sigue las tendencias marcadas por el segundo bloque. De este modo, a partir de los ocho hilos la ganancia es pobre. Esta adaptación debería utilizarse solamente en sistemas con un nivel de paralelismo bajo o moderado para no desaprovechar las capacidades de la máquina.

Por último, la versión combinada es la que mejor escala con el número de *threads* utilizados, debido a que no está limitada a paralelizar ocho soluciones parciales por una parte, y a que, al dedicar ocho veces menos hilos al cálculo de cada una de las susodichas soluciones parciales respecto a la estrategia de franjas, el desbalanceo no se produce hasta cantidades más elevadas de hilos. Por ello, la suavización de la ganancia que ofrece respecto al número de hilos utilizados probablemente tenga como causa el que se haya alcanzado el límite de carga del procesador para todos los núcleos. Podemos ver que, para las últimas mediciones, supera a las otras dos aproximaciones pero que, en cualquier caso, el máximo de rendimiento se obtiene con la solución de franjas para una cantidad más reducida de *threads*, por lo que parece que, incluso para un sistema tan paralelo como este para el que había sido diseñada esta versión, no aporta una ganancia suficiente respecto a las otras para ser útil. Podría pasar, no obstante, que en un sistema con capacidad para correr más hilos, la escalabilidad de este método sí lo convierta en el más adecuado. Por ello, parece que debería utilizarse únicamente en arquitecturas extremadamente paralelas.

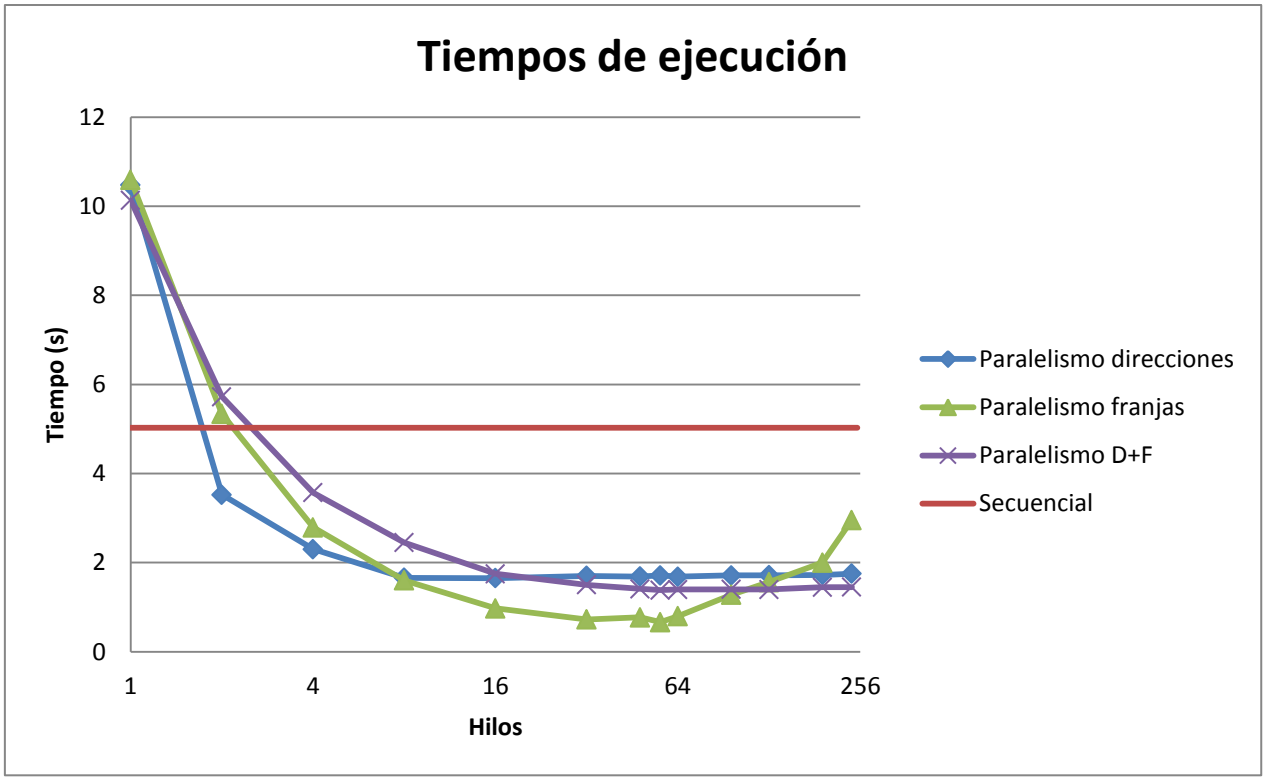
A continuación, realizaremos el desglose por áreas ya visto en el apartado anterior para este algoritmo, ilustrado mediante las Gráficas 8.24, 8.25, 8.26, 8.27, 8.28 y 8.29.



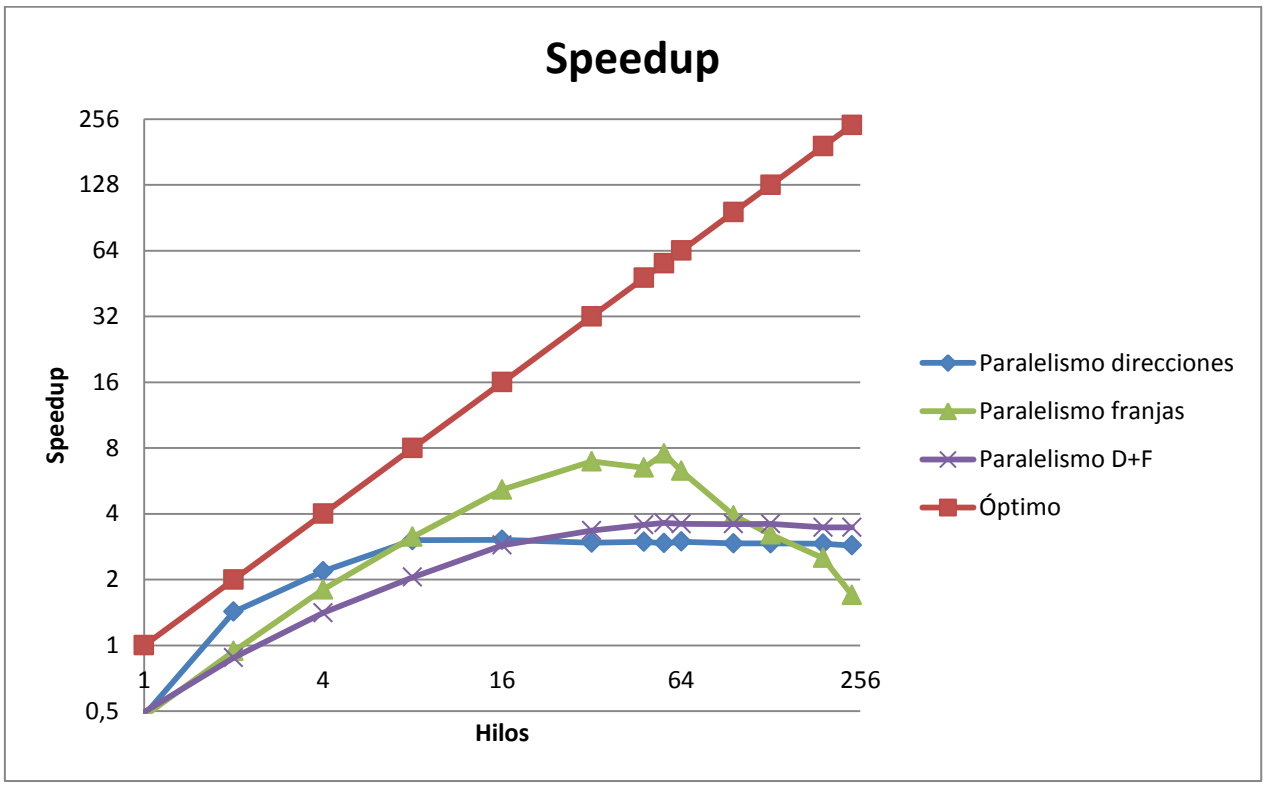
GRÁFICA 8.24: TIEMPOS DE EJECUCIÓN PARA SEMI-GLOBAL MATCHING (BLOQUE 1), PHI



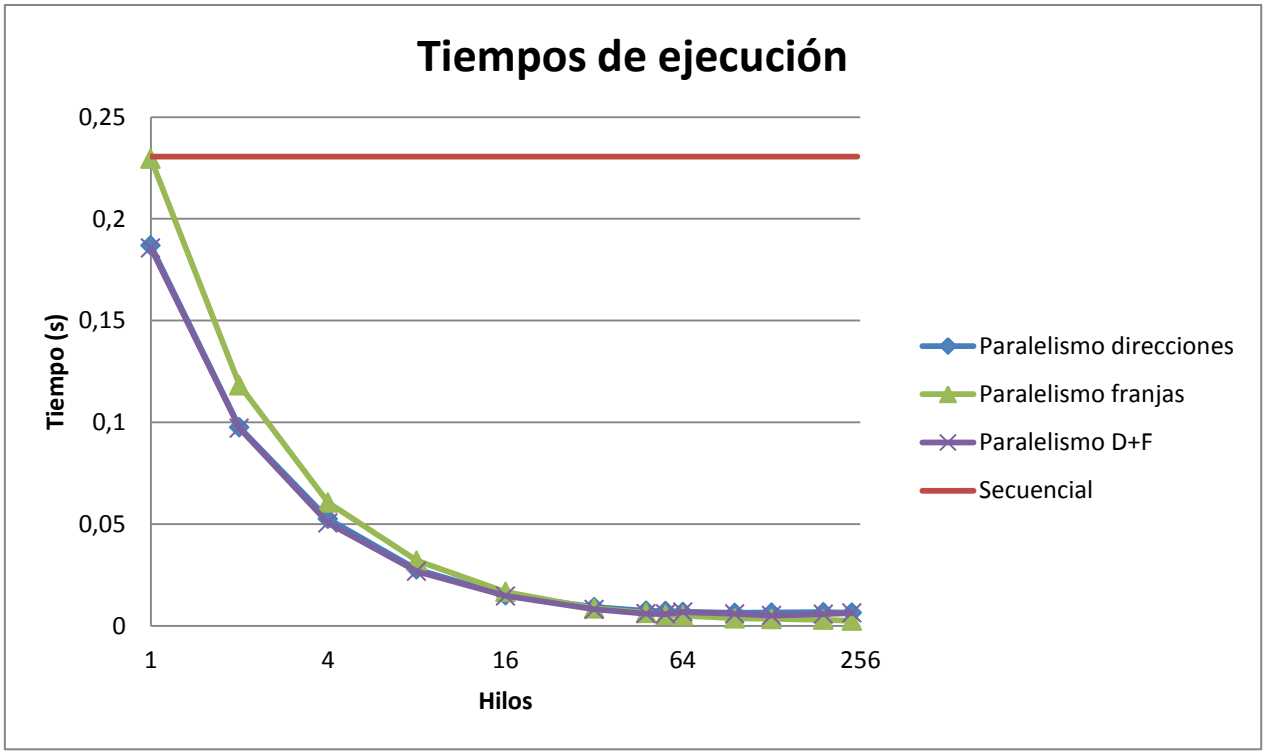
GRÁFICA 8.25: SPEEDUP PARA SEMI-GLOBAL MATCHING (BLOQUE 1), PHI



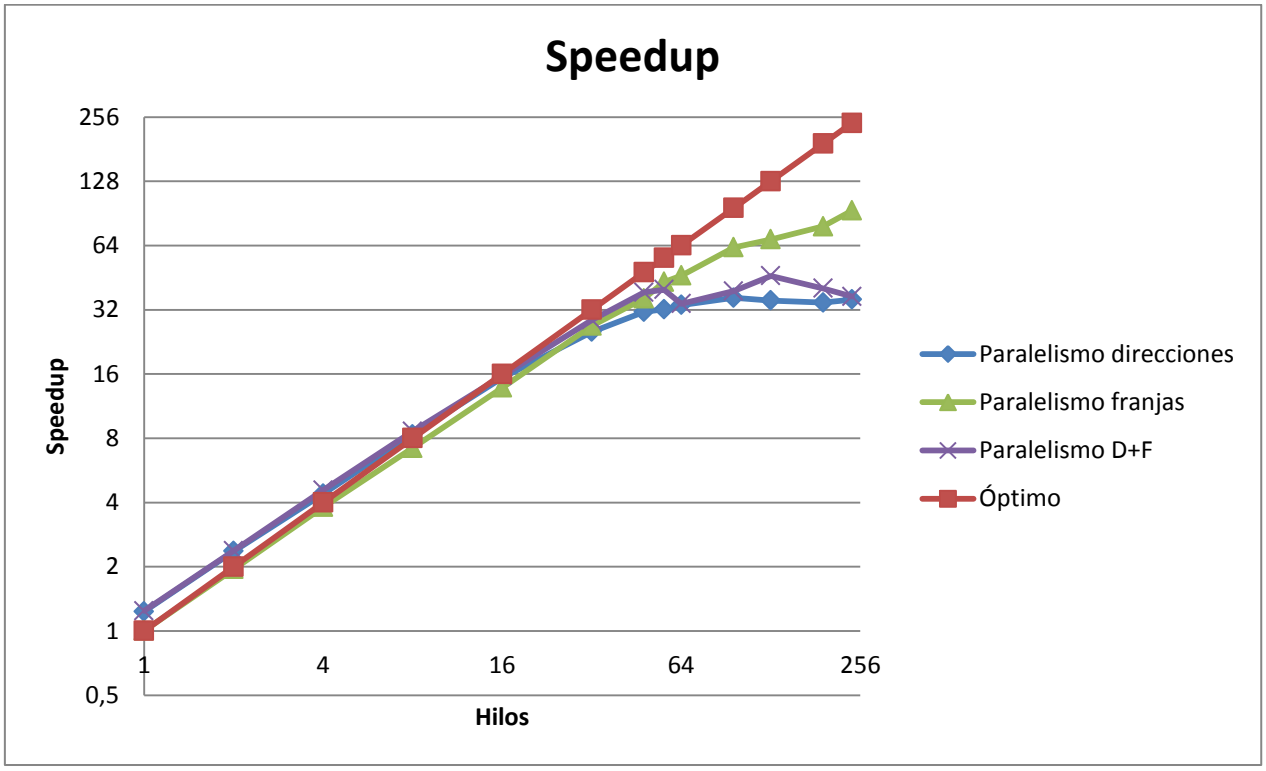
GRÁFICA 8.26: TIEMPOS DE EJECUCIÓN PARA SEMI-GLOBAL MATCHING (BLOQUE 2), PHI



GRÁFICA 8.27: SPEEDUP PARA SEMI-GLOBAL MATCHING (BLOQUE 2), PHI



GRÁFICA 8.28: TIEMPOS DE EJECUCIÓN PARA SEMI-GLOBAL MATCHING (BLOQUE 3), PHI



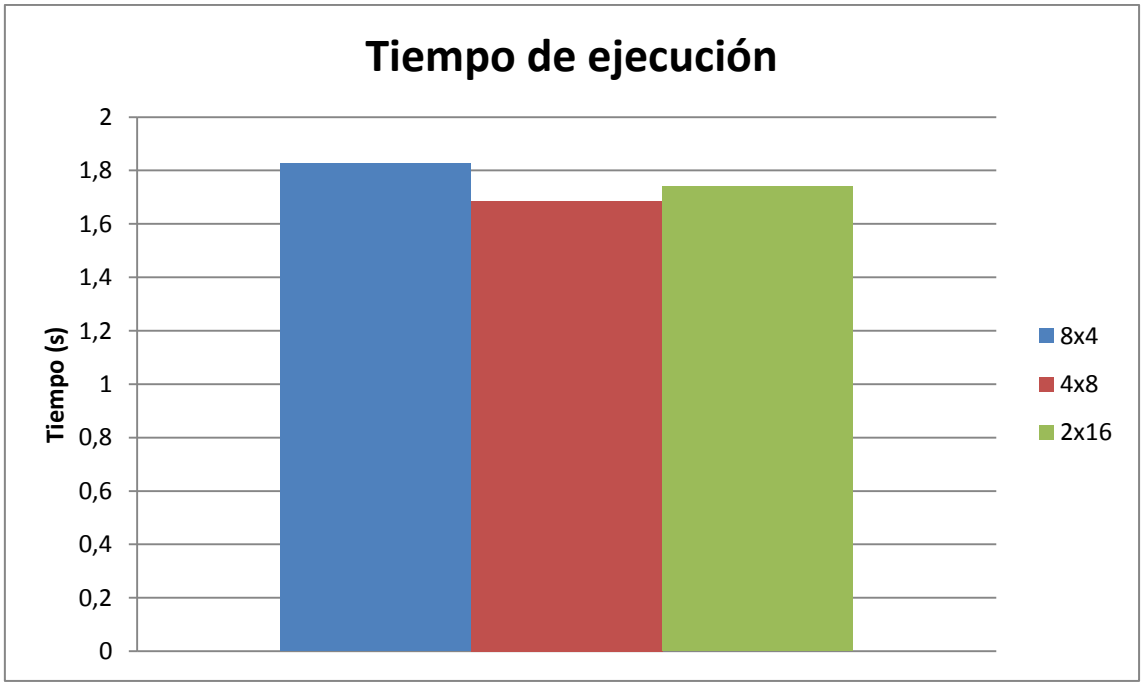
GRÁFICA 8.29: SPEEDUP PARA SEMI-GLOBAL MATCHING (BLOQUE 3), PHI

Como se puede observar, en los bloques primero y tercero escala mejor el rendimiento que en la arquitectura anterior, lo que probablemente sea debido al hecho ya mencionado de que el procesamiento secuencial en esta es más costoso y,

por lo tanto, es más probable que se compense el coste de creación de los hilos con la ganancia que aportan. Se ve, también, que el comportamiento en estos bloques se ve afectado por la ejecución del resto del programa en un grado claramente inferior, ya que el comportamiento es el esperado para estas regiones: que las diferentes estrategias tengan rendimientos parejos debido a que la paralelización de estas secciones es la misma para todas, y que la escalada de rendimiento se acerque razonablemente al óptimo, mucho más en el tercer bloque porque los hilos ya están creados y las cachés con datos útiles, y no hay que compensar este coste.

Respecto al segundo bloque, como cabía esperar, marca las tendencias que hemos visto en los tiempos de ejecución de la totalidad del algoritmo, de modo que la justificación para estas ya se ha propuesto al comienzo de este subapartado.

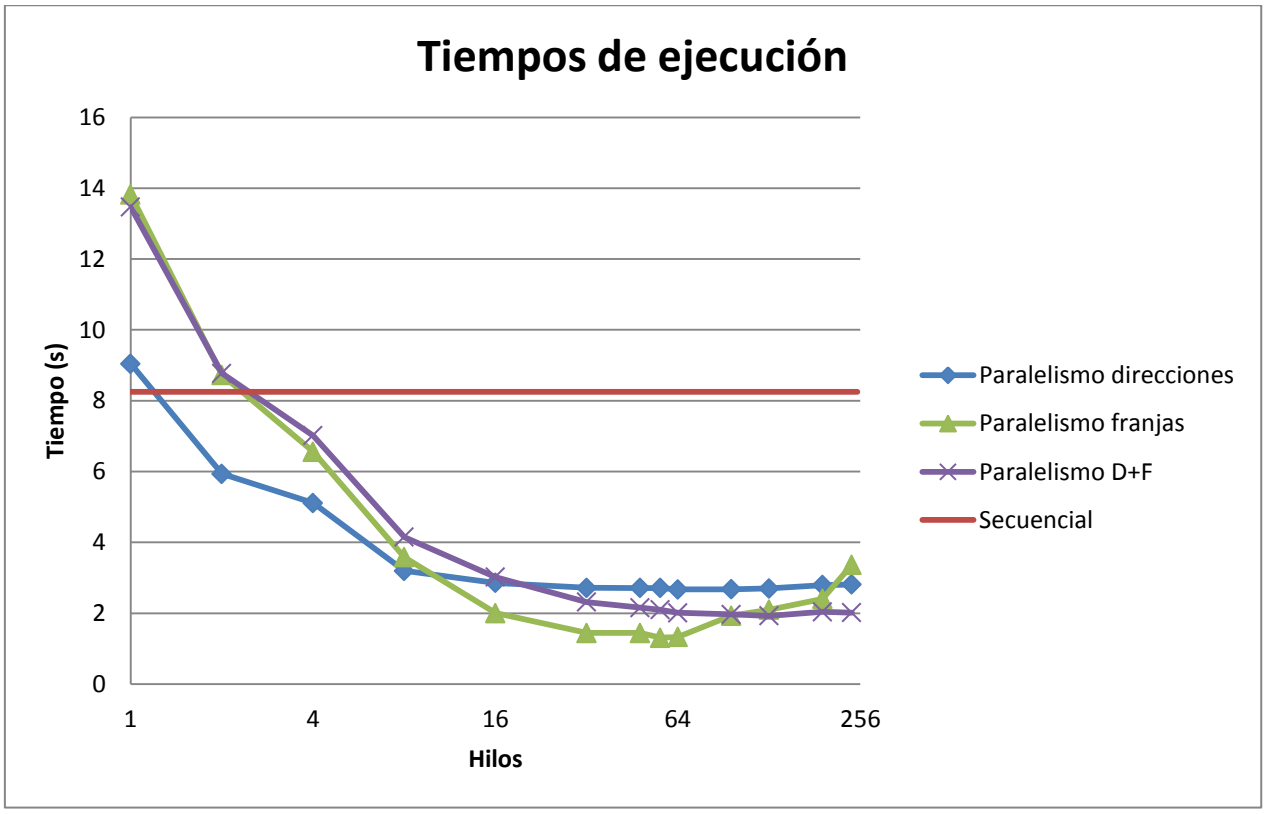
Profundizando en el comportamiento de este algoritmo, volvemos a realizar sobre esta arquitectura el mismo experimento visto en la anterior: lanzar la estrategia de paralelización combinada con diferentes repartos de hilos de ejecución entre el lazo externo y el interno. En este caso se prueba para 32 *threads* con la versión genérica, que usa 8 para el exterior y 4 para el interior; y con otras dos con 4 para el exterior y 8 para el interior, y 2 para el exterior y 16 para el interior, respectivamente. Los resultados se pueden ver en la siguiente gráfica de barras:



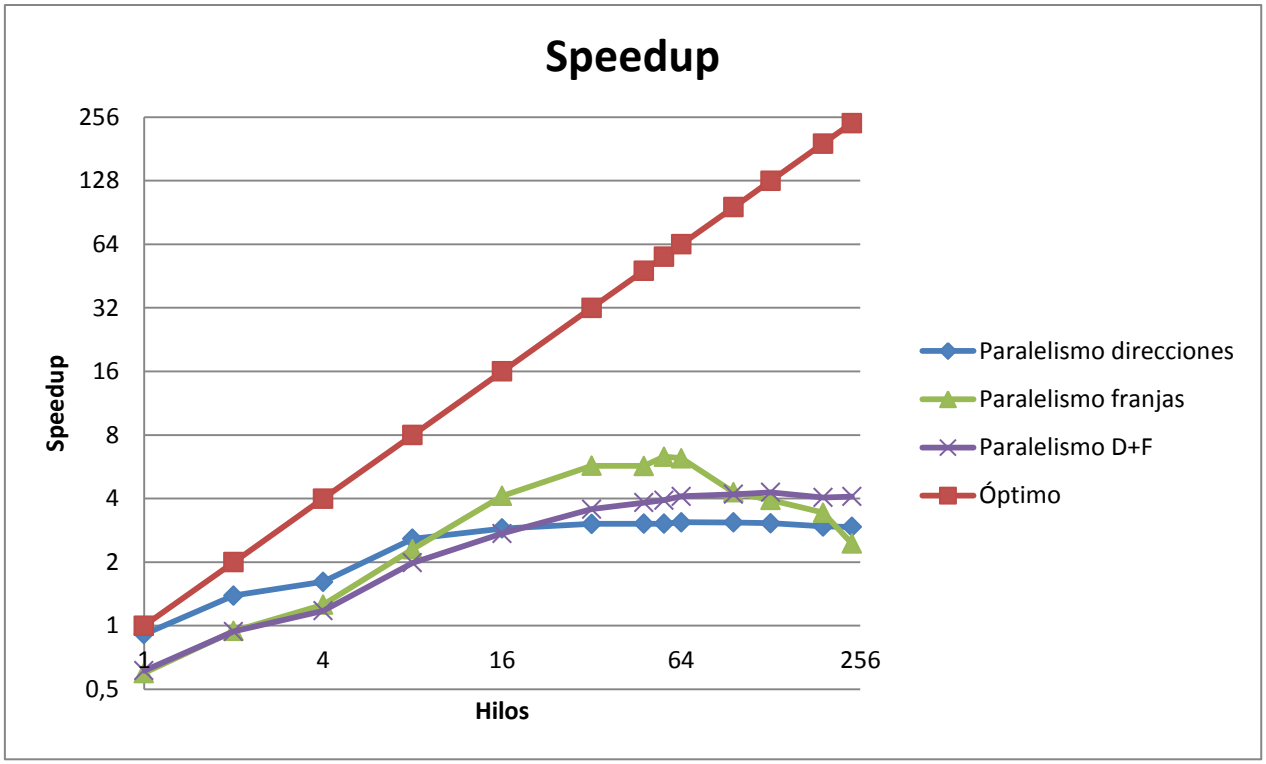
GRÁFICA 8.30: COMPARATIVA DE DIFERENTES REPARTOS ENTRE LAZOS EXTERNO E INTERNO, PHI

Una vez más, se observa que las diferencias entre las diferentes aproximaciones al reparto de la carga de trabajo son poco significativas respecto al tiempo de ejecución del programa, por lo que parece un aspecto menor su elección a la hora de optimizar el rendimiento del algoritmo.

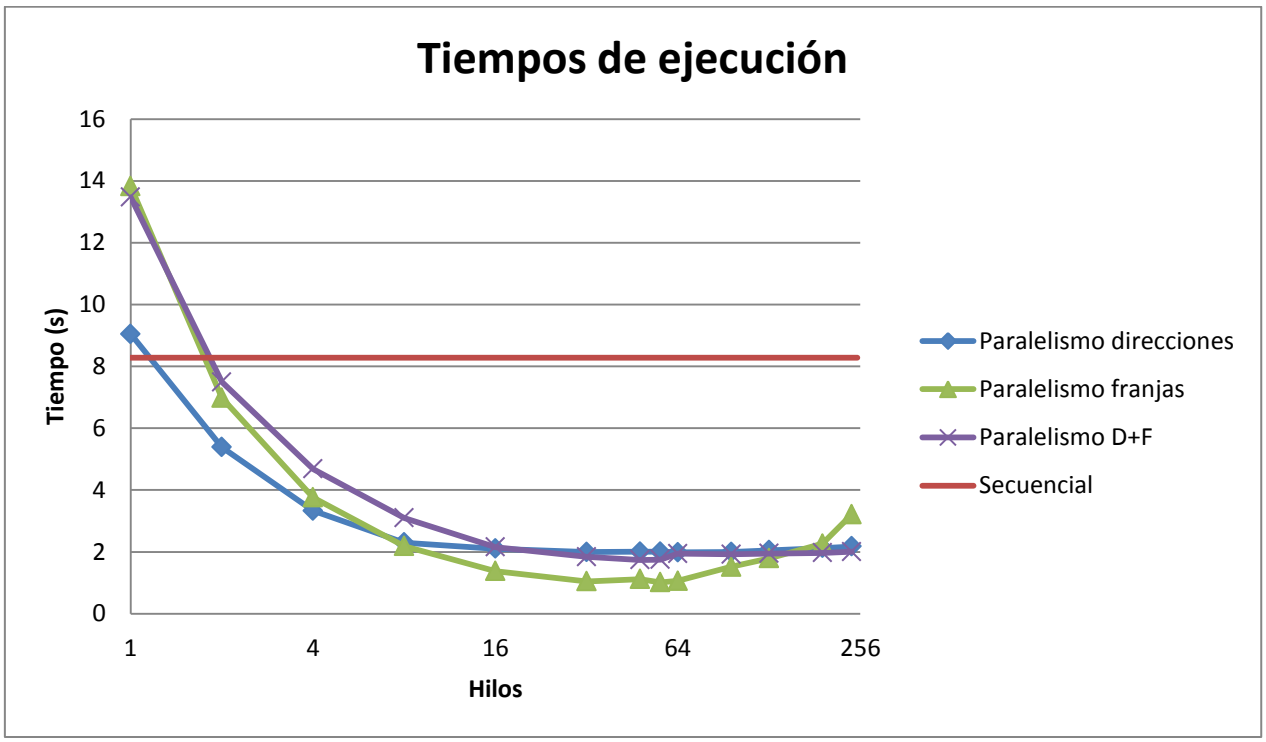
Por último, en este subapartado también se hará un análisis comparativo, como ya se había anunciado anteriormente, sobre el uso de los diferentes tipos de afinidades para la distribución de hilos en los núcleos. Recordamos que las opciones que nos brinda el compilador de Intel son *scattered* (la utilizada por defecto), *compact* y *balanced*. A estos efectos, incluimos a continuación las Gráficas 8.31, 8.32, 8.33 y 8.34 con el rendimiento de la ejecución del total del algoritmo para las dos aproximaciones restantes.



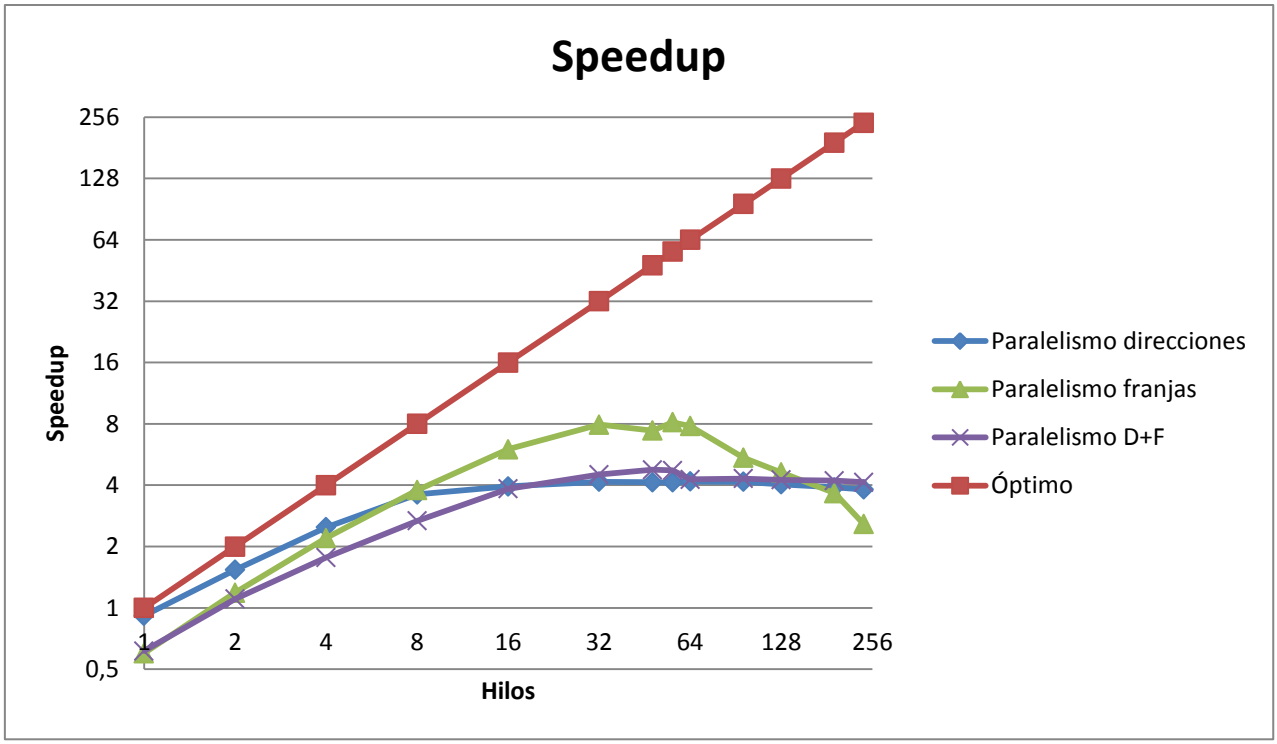
GRÁFICA 8.31: TIEMPOS DE EJECUCIÓN PARA SEMI-GLOBAL MATCHING, COMPACT



GRÁFICA 8.32: SPEEDUP PARA SEMI-GLOBAL MATCHING, COMPACT



GRÁFICA 8.33: TIEMPOS DE EJECUCIÓN PARA SEMI-GLOBAL MATCHING, BALANCED



GRÁFICA 8.34: SPEEDUP PARA SEMI-GLOBAL MATCHING, BALANCED

Respecto al modo *compact*, podemos ver que se comporta de forma similar a la medida hasta ahora, con dos diferencias principales: la ganancia es, en general, peor; y existen pequeños saltos en la pendiente de esta frente al número de hilos de ejecución utilizados. Ambas diferencias se deben al hecho de que se rellenan todos los hilos de un núcleo antes de pasar al siguiente, como habíamos visto ya en el capítulo que describía las arquitecturas consideradas. De este modo, se fuerza el uso del paralelismo de hilos mediante la tecnología Hyper-Threading mientras sea posible, en lugar de repartir los hilos entre los distintos núcleos físicos. Como es natural, este pseudo-paralelismo, a pesar de ser más eficiente que una ejecución secuencial en la mayoría de los casos, no lo es tanto como el paralelismo real que ofrecen los núcleos independientes. Por ello, el rendimiento es peor, en general, que el obtenido mediante la estrategia de reparto *scattered*, y se producen estos pequeños saltos coincidentes con la utilización de un núcleo adicional cuando se llena el actual.

Respecto al esquema *balanced*, podemos ver que el comportamiento de los resultados de rendimiento del algoritmo cuando se aplica son prácticamente idénticos a los analizados a lo largo de este capítulo para el modo *scattered*. Por lo tanto, parece que el hecho de que hilos con números contiguos se asignen al mismo núcleo no tiene un efecto relevante para la ejecución de este algoritmo.

8.3. Comparación de los resultados para ambas arquitecturas

En este apartado, se realizará una breve comparación de los resultados obtenidos por cada arquitectura, que complementará todo lo que ya se ha dicho al respecto a lo largo de este capítulo. Servirá, pues, como conclusión de este.

En primer lugar, lo más evidente es que para ejecuciones con números de hilos reducidos, el cluster ofrece resultados claramente superiores, como cabe esperar dado que sus núcleos tienen prestaciones sensiblemente mejores que los del coprocesador. Dejando esto a un lado, nos adentraremos en los comportamientos de cada uno de los algoritmos en términos de escalabilidad a medida que se aumenta el número de hilos.

En el *Pixel-wise Matching*, podemos ver que la escalada de rendimiento es mejor en el servidor, sin llegar a decaer tanto como en el coprocesador, y alcanzando un tiempo mínimo de ejecución de orden diez veces inferior al obtenido en la otra arquitectura. En este caso, para un procesamiento tan breve, podemos ver que el alto nivel de paralelismo del Xeon Phi resulta fútil.

En el caso del *Patch-wise Matching*, podemos ver que ambas versiones (*Sum of Absolute Differences* y *Adaptive Support-Weight*) escalan mejor en el coprocesador, debido a su prolongado tiempo de cálculo y a lo sencillo y equitativo que es el reparto de las cargas de trabajo, acercándose mucho a la ganancia ideal hasta cantidades de hilos elevadas. No obstante, solamente en la segunda variación se obtiene un mínimo para el tiempo total de cómputo inferior al obtenido por el cluster, al alcanzar un factor de ganancia mayor que 84. Los mínimos en este caso son 2,06s y 3,17s, respectivamente, lo que significa que en el Phi se obtiene una ventaja de aproximadamente el 35% para la mejor ejecución de esta versión. Para la otra, los mejores resultados son 0,99s y 0,56s, respectivamente, es decir, la mejor ejecución del coprocesador consume de forma aproximada un 177% del tiempo utilizado por la mejor ejecución en el servidor. Con esto, podemos ver que el paralelismo que ofrece el Xeon Phi solamente resulta útil con cargas de trabajo muy elevadas, como la que supone la segunda variación de este algoritmo.

Por último, respecto al *Semi-Global Matching*, volviendo a un algoritmo con una carga de cómputo reducida, podemos ver que, a pesar de que la escalada de rendimiento es mucho mejor en el coprocesador, llegando a factores de ganancia superiores a ocho para el método por franjas frente a máximos inferiores a dos en el servidor, en el primero no se alcanzan tiempos de ejecución tan buenos como aquéllos obtenidos en el segundo. Esto se debe, como ya se ha explicado, a que para cargas de trabajo tan reducidas, el alto nivel de paralelismo no compensa el hecho de que las prestaciones de los núcleos del coprocesador sean acusadamente inferiores a las que tienen los del cluster. De este modo, las mejores ejecuciones con la estrategia que ofrece el mayor rendimiento en el Xeon Phi solamente consiguen aproximarse a tiempos del orden de 1s, sin llegar a alcanzar este, mientras que el procesador

principal logra un mínimo de casi 0,4s, aproximadamente un 60% menos. Estos mínimos se obtienen, en ambos casos, por medio de la estrategia de franjas.

Como conclusión a este apartado cabe decir que, como sugiere su diseño, el uso del coprocesador solamente es aconsejable frente al del procesador principal para cargas de trabajo muy elevadas y con alta escalabilidad respecto al número de hilos utilizados. Hay que tener en cuenta también, que en este caso la comparación se realiza entre un cluster con dos procesadores principales frente a solamente un coprocesador, de modo que para el caso con un solo procesador principal, la carga de trabajo necesaria para que compense la utilización del Xeon Phi debería ser considerablemente inferior.

9. CONCLUSIONES

Las potenciales utilidades del Stereo Matching requieren cada vez un rendimiento de ejecución más elevado, ya que se tiene a aplicaciones portátiles con requerimientos de respuestas en tiempo real, como puede ser la robótica; o incluso otras destinadas a manejar grandes cantidades de datos, como la reconstrucción tridimensional de escenarios. Por todo esto, resulta muy importante la optimización de los algoritmos de este campo.

En este trabajo de fin de grado se ha planteado el estudio de algunos de los algoritmos de los que conforman el estado del arte en esta disciplina, llevando a cabo su implementación secuencial y añadiendo mejoras basadas en la paralelización, cuyo rendimiento ha sido testeado en dos arquitecturas muy diferentes, sobre las que se han realizado mediciones.

Las mediciones realizadas han sido analizadas desde diferentes enfoques comparativos: la escalabilidad de rendimiento de cada algoritmo en sus versiones paralelas respecto a la secuencial, la ejecución de estos en ambas arquitecturas o el estudio de parámetros que afectan a la paralelización, como las afinidades en la distribución de hilos entre los núcleos, o el reparto de estos entre dos niveles de anidamiento de la paralelización.

Estos resultados arrojan algunos datos de interés científico, como el hecho de que la ganancia de rendimiento basada en el incremento de hilos de ejecución es muy pobre para cargas de trabajo reducidas en estas arquitecturas. Otro ejemplo es el hecho de que la utilidad del coprocesador frente al servidor utilizado se limita a tareas con cargas computacionales muy elevadas y altamente escalables respecto al rendimiento frente al número de hilos utilizados (una vez más, hay que tener en cuenta que el servidor incluye dos procesadores CPUs).

Además del aprendizaje que han facilitado los resultados, se han conseguido tasas de mejora de rendimiento muy importantes en algunos casos llegando, como dato reseñable, a obtener en una paralelización un factor de *speedup* superior a 84 respecto a la versión secuencial. En las variantes de *Patch-wise Matching* es donde se ha obtenido los efectos más positivos con las ejecuciones paralelas, con ganancias más moderadas para el *Pixel-wise Matching* por su bajo coste computacional base, que hace que no sea rentable un nivel elevado de paralelización; y el *Semi-Global Matching*, cuya compleja paralelización obliga, para su implementación, a alejarse de las tasas de ganancia ideales en base al número de *threads* utilizados.

El alcance de este proyecto, no obstante, está limitado por el plazo de entrega, con lo que se ha decidido dejar fuera de éste algunas actividades que podrían plantearse como una ampliación del estudio de cara al futuro.

Por un lado, el estado del arte de esta disciplina abarca una gran diversidad de algoritmos, cada uno con sus características particulares que pueden favorecer su uso en una u otra situación. Por ello, cabría realizar una ampliación añadiendo algunos de ellos con características diferentes a los ya considerados.

También se podrían plantear más estrategias de paralelización para los casos no triviales, que en este proyecto se reducen al *Semi-Global Matching*, que se podría haber abordado, por ejemplo, mediante una estrategia de paralelización por *tiling* [20], para realizar una comparativa más amplia y ver qué método se adapta mejor a las características del algoritmo. Del mismo modo, se podrían utilizar las instrucciones de vectorización propias del Xeon Phi.

Otro punto de ampliación sería la inclusión de más arquitecturas en el estudio, especialmente de la placa *Parallella*, cuya utilización para la realización de medidas se ha descartado por problemas técnicos y falta de documentación para su uso. Esto sería muy interesante, dado que se evaluarían las soluciones paralelas en un microprocesador empotrado, adaptándose a las mencionadas tendencias actuales en el campo del Stereo Matching.

Por último, cabría considerar la utilización de imágenes mayores para estudiar el comportamiento de los algoritmos con volúmenes de datos más cuantiosos, que cabe esperar que sean diferentes a los observados a lo largo de este proyecto: se esperaría encontrar unos resultados de escalabilidad mejores. En este caso, esta ampliación se ha dejado fuera del alcance debido a que, en la fuente utilizada para la obtención de imágenes de muestra [8], no se proporcionan mapas de oclusión para las últimas remesas, y su cálculo resulta demasiado complejo como para ser abordado en este trabajo de fin de grado.

APÉNDICE A - MANUAL DE USUARIO

En este apartado se describen los pasos necesarios para la ejecución de este software en Linux (probado para las distribuciones Ubuntu 14.04 y Centos 6.6), tanto desde línea de comandos como por medio de la interfaz gráfica disponible. No se incluirán capturas de pantalla, ya que estas pueden verse en la Sección 4, correspondiente a la arquitectura del software desarrollado.

El software se encuentra en su versión final, lista para el uso, en un repositorio privado del sitio web BitBucket, con lo que es necesaria una invitación para su obtención a través de GIT. También se puede obtener como una copia local de dicho repositorio comprimida en formato ZIP. En cualquier caso, desde esta configuración el software está listo para su compilación y posterior uso. La compilación puede realizarse a través de la herramienta *make*, utilizando el *makefile* que se incluye; o bien directamente llamando al compilador elegido teniendo en cuenta que, dependiendo de cuál sea, podría resultar necesario el uso de *flags* para cargar las librerías de matemáticas de C y OpenMP. Un ejemplo para ICC sería:

```
icc main.cpp –openmp –o stereo
```

Esto resultaría en un ejecutable llamado “stereo” listo para su uso.

A.1. Ejecución por línea de comandos

Para la ejecución de este software a través de la línea de comandos, debe abrirse una terminal, situada en el directorio raíz donde está el ejecutable, y lanzarlo con la siguiente sintaxis:

```
./<ejecutable> <img1> <img2> <img_result> [-r] –o <output_range> -O
<output_scale> [-a <action>] [-W <patch_width>] [-H <patch_height>] [-
t <comparison_type>] [-p <parallel_mode> [-n <number_of_threads>]] [-w
<results_file> -g <ground_truth> -m <occlusion_map>]
```

Donde los argumentos entre corchetes son opcionales, y los identificadores entre picos significan lo siguiente:

- img1: Imagen base para el procesamiento.
- img2: Imagen secundaria para el procesamiento.
- img_result: Ruta donde se quiere almacenar el mapa de profundidad resultante.
- output_range: Rango de búsqueda pre-calculado en el que se deben buscar los posibles candidatos para el emparejamiento con cada píxel base.
- output_scale: Factor de escala por el que se multiplicará cada píxel del mapa de profundidad resultante de la ejecución del algoritmo para que cubra toda la escala de grises. Depende de <output_range>.

- action: Acción que se quiere que realice el programa. Puede ser:
    - 0 para ejecución de *Pixel-wise Matching* (opción por defecto)
    - 1 para ejecución de *Patch-wise Matching*
    - 2 para ejecución de *Semi-Global Matching*
- patch_width: Ancho del parche que se utilizará. Solamente tiene efecto si la acción elegida es la ejecución de *Patch-wise Matching*.
- patch_height: Alto del parche que se utilizará. Solamente tiene efecto si la acción elegida es la ejecución de *Patch-wise Matching*.
- comparison_type: Tipo de comparación realizada. Solamente tiene efecto si la acción elegida es la ejecución de *Patch-wise Matching*. Puede ser:
    - 0 para comparación basada en *Sum of Absolute Differences* (opción por defecto)
    - 1 para comparación basada en *Adaptive Support-Weight*
- parallel_mode: Si no se indica, se lanza la versión secuencial por defecto. Si se indica, selecciona el modo de paralelismo empleado en la ejecución, que puede ser:
    - Para *Pixel-wise Matching*:
        - 0 para paralelismo básico
    - Para *Patch-wise Matching*:
        - 0 para paralelismo básico
    - Para *Semi-Global Matching*:
        - 0 para paralelismo por direcciones
        - 1 para paralelismo por franjas
        - 2 para paralelismo combinado por direcciones y franjas
- number_of_threads: Número de hilos utilizado en una ejecución paralela. El valor por defecto es 8. Si la ejecución del algoritmo es secuencial, este valor se ignora.
- results_file: Ruta donde se guardará el archivo con los resultados, o donde se añadirán si el archivo ya existe. Por defecto no se guardan los resultados.
- ground_truth: Mapa de profundidad que se utilizará como *ground truth* para evaluar la precisión obtenida en la ejecución de un algoritmo sobre un par de imágenes.
- occlusion_map: Mapa de píxeles ocluidos que se utilizará para evaluar la precisión obtenida en la ejecución de un algoritmo sobre un par de imágenes.

Además de esto, con la opción `-r` se indica que la imagen base es la derecha, con lo que los algoritmos utilizados se adaptarán a esta condición para la búsqueda de candidatos de emparejamiento para los píxeles base.

Cualquier llamada al ejecutable con unos argumentos que no se adapten a la sintaxis resultará en que el software muestre una pantalla de ayuda aclaratoria.

A.2. Ejecución mediante interfaz gráfica

Los parámetros que el front-end gráfico pide para su ejecución ya están explicados en el punto anterior, ya que son un subconjunto de los pedidos por línea de comandos. Por este motivo, y por el hecho de que la interacción con una interfaz gráfica es muy visual e intuitiva, no se explica nada más al respecto en este punto.

Para ejecutar este front-end, únicamente es necesario tener una instalación de la máquina virtual de Java en el sistema, copiar el archivo .jar en el mismo directorio que el ejecutable, que deberá llamarse “stereo”, y lanzarlo. Esto se puede hacer desde una terminal con el siguiente comando:

```
java –jar Front_end.jar
```

Una vez abierta la ventana, simplemente han de cubrirse los campos en base a lo indicado anteriormente y hacer clic en el botón “Run test”, que lanzará el ejecutable pasando los parámetros establecidos por línea de comandos, y esperará a que termine su procesamiento. Una vez finalizado, pasará a un panel donde se muestran la imagen base y el mapa de profundidad generado para ella, ambas recortadas para encajar en los cuadros dispuestos para su visualización, siempre respetando las proporciones originales. Además, se mostrarán los resultados de rendimiento y precisión en modo texto en el cuadro inferior, si es que se ha elegido la opción de almacenar dichos resultados. En otro caso, se mostrará un mensaje que indica que no se ha seleccionado la mencionada opción. Por último, en la parte inferior hay un botón, cuyo contenido textual es “Return to form”, que permite al usuario volver al formulario anterior sin haber borrado sus campos, para realizar posteriores ejecuciones con los mismos u otros parámetros.

BIBLIOGRAFÍA

[1] R. Hartley y A. Zisserman, «Epipolar Geometry and the Fundamental Matrix,» de *Multiple View Geometry in Computer Vision*, Cambridge University Press, 2004, pp. 239-261.

[2] H. Hirschmüller, «Accurate and Efficient Stereo Processing by Semi-Global Matching and Mutual Information,» *Conference on Computer Vision and Pattern Recognition,* vol. 2, pp. 807-814, 2005.

[3] T. Kanade, «Development of a video-rate stereo machine.,» de *Image Understanding Workshop*, Monterey, CA, Morgan Kaufmann Publishers, 1994, pp. 549-557.

[4] S. Mattocia, «Department of Computer Science and Engineering (DISI), University of Bologna, Italy,» 20 04 2010. [En línea]. Available: http://vision.deis.unibo.it/~smatt/Seminars/StereoVision.pdf. [Último acceso: 20 06 2015].

[5] Intel, «Intel Xeon,» [En línea]. Available: http://ark.intel.com/products/75791. [Último acceso: 22 06 2015].

[6] Intel, «Intel Xeon Phi,» [En línea]. Available: http://ark.intel.com/products/75791. [Último acceso: 22 06 2015].

[7] Adapteva, «Parallella,» [En línea]. Available: https://www.parallella.org/board/. [Último acceso: 22 06 2015].

[8] Middlebury College, «Middlebury Stereo,» [En línea]. Available: http://vision.middlebury.edu/stereo/. [Último acceso: 20 02 2015].

[9] K. Beck, Extreme Programming Explained: embrace change, Adison-Wesley Proffessional, 2000.

[10] Atlassian Software, «BitBucket,» [En línea]. Available: https://bitbucket.org/.

[11] Google, «Drive,» [En línea]. Available: https://drive.google.com.

[12] InfoJobs, «InfoJobs,» [En línea]. Available: https://www.infojobs.net/. [Último acceso: 29 06 2015].

[13] Job and Talent, «Job and Talent,» [En línea]. Available:

http://www.jobandtalent.com/es. [Último acceso: 29 06 2015].

[14] Universidade de Santiago de Compostela, «USC Investigación,» 11 11 2008. [En línea]. Available: http://imaisd.usc.es/ftp/oit/documentos/591_gl.pdf. [Último acceso: 21 3 2015].

[15] OpenMP Architecture Review Board, «OpenMP,» [En línea]. Available: http://openmp.org/wp/about-openmp/. [Último acceso: 17 06 2015].

[16] K.-J. Yoon y S.-K. In, «Adaptive Support-Weight Approach for Correspondence Search,» de *IEEE transactions on Pattern Analysis and Machine Intelligence*, 2006.

[17] Intel, «Intel Hyper-Threading,» [En línea]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html. [Último acceso: 22 06 2015].

[18] J. Jeffers y J. Reinders, de *Intel Xeon Phi Coprocessor High Performance Programming*, Waltham, MA, Morgan Kaufmann, 2013, pp. 189-241.

[19] Intel, «Intel C++ Compiler XE 13.1 user and Reference Guides,» [En línea]. Available: https://software.intel.com/sites/products/documentation/doclib/iss/2013/compiler/cpp-lin/GUID-8FCD3720-6F73-429C-AE65-7144ED0B991A.htm. [Último acceso: 22 06 2015].

[20] J. Xue, «Loop Tiling for Parallelism,» *Kluwer Academic Publishers,* 2000.

[21] D. Scharstein y R. Szeliski, «A taxonomy and evaluation of dense two-frame stereo correspondence algorithms,» *International Jorunal of Computer Vision,* nº 47, pp. 7-42, 2002.