Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

"Prestaciones y energía de esquemas de mapeo sin precarga para aceleradores CNN sistólicos"

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Eduardo Yago Vicent

Tutores: Julio Sahuquillo y
María Engracia Gómez Requena

2018-2019

Agradecimientos

En primer lugar, me gustaría agradecer a mis tutores la oportunidad que me brindaron al proponerme la participación y desarrollo del proyecto sobre el que se basa este trabajo y también agradecer los conocimientos que he adquirido trabajando con ellos.

En segundo lugar, me gustaría agradecer a mi compañero Pau Castelló Ferrer por su colaboración conjunta en la investigación del proyecto.

En tercer lugar, agradecer a todos los profesores que a lo largo de estos años de estudio me han ayudado a adquirir los conocimientos y habilidades que han hecho posible que sea capaz de desarrollar este trabajo.

Para terminar, me gustaría dar las gracias a mi familia que durante todos estos años de estudios han hecho posible que yo haya llegado hasta aquí y ya que, sin su apoyo, ni mis estudios ni este trabajo habrían sido posibles.

Abstract

There are several types of neural networks depending on their characteristics. This study focuses on a specific type of neural network called Convolutional Neural Networks (CNN). The high relevance of this type of network, due to its application in image recognition or the field of artificial intelligence, has led to a strong interest by the scientific community in developing new mechanisms to increase efficiency and computing power of such neural networks. One of the proposals that are gaining the most strength is the development of systolic or purpose-built processors. Examples include Google's recent TPU [9] or Eyeriss [10].

This work focuses on using an array simulator or systolic processor called SCALE-Sim for CNN computation to identify critical parameters in the hardware architecture of such devices and then propose concrete changes to achieve greater efficiency, maximizing the ratio of productivity, utilization and energy consumption.

To do this, first of all, a detailed study of the architecture of the simulator itself is made, identifying its parameters and understanding its operation. The simulator reads some configuration files for execution and generates some output files with the results about the behavior and performance that the configured array would have had when running a certain neural network. Understanding the input files and especially the output files is key to then analyze the results and try to optimize the configurations.

After the study stage of the simulator with which we work, we proceed to the stage of experimentation, choosing a finite set of CNN but varied making sure that it is as representative as possible and then proceeds to do experiments by changing different parameters in the architecture of the hardware being simulated to study the impact it would have on performance and consumption.

After this stage, we proceed to analyze the results obtained and the derivation of the results obtained directly from the SCALE-Sim in additional results such as energy consumption. To do this, the SCALE-Sim output data is used to calculate the energy data using the CACTI simulator.

Once all the results have been analyzed, we proposed optimizations in the configurations of the architecture.

Resumen

Existen varios tipos de redes neuronales según sus características. Este estudio se centra en un tipo específico de red neuronal llamado Convolutional Neural Networks (CNN). La gran relevancia de este tipo de redes, debido a su aplicación en el reconocimiento de imágenes o en el campo de la inteligencia artificial, ha desembocado en un fuerte interés por la comunidad científica en desarrollar nuevos mecanismos para aumentar la eficiencia y el poder de cómputo de este tipo de redes neuronales. Una de las propuestas que más fuerza está cobrando es el desarrollo de procesadores sistólicos o de propósito específico. Ejemplos de ello son la reciente TPU de Google [9] o el Eyeriss [10].

Este trabajo se centra en el uso de un simulador de array o procesador sistólico llamado SCALE-Sim para el cómputo de CNN para identificar parámetros críticos en la arquitectura hardware de este tipo de dispositivos para posteriormente proponer cambios concretos en su configuración con el fin de obtener una mayor eficiencia, maximizando la ratio entre productividad, utilización y consumo energético.

Para ello primeramente se hace un estudio detallado de la arquitectura del propio simulador, identificando sus parámetros y comprendiendo su funcionamiento. El simulador lee unos ficheros de configuración para su ejecución y genera unos archivos de salida con los resultados sobre el comportamiento y rendimiento que habría tenido el array configurado al ejecutar una determinada red neuronal. La comprensión de los ficheros de entrada y sobre todo los de salida es clave para después analizar los resultados y tratar de optimizar las configuraciones.

Después de la etapa del estudio del simulador con el que se trabaja, se procede a la etapa de experimentación, eligiendo un conjunto finito de CNN, pero variado procurando que sea lo máximo representativo posible y se procede a hacer ensayos cambiando distintos parámetros en la arquitectura del hardware que se está simulando para estudiar el impacto que tendría en las prestaciones y consumo.

Después de esta etapa se procede a analizar los resultados obtenidos y la derivación de los resultados que se obtienen directamente del SCALE-Sim en resultados adicionales como es el consumo energético. Para ello en el estudio se utilizan los datos de salida del SCALE-Sim para calcular los datos energéticos mediante el simulador CACTI.

Una vez analizados todos los resultados se proponen optimizaciones en las configuraciones de la arquitectura simulada.

Resum

Hi ha diversos tipus de xarxes neuronals segons les seues característiques. Este estudi se centra en un tipus específic de xarxa neuronal cridat Convolutional Neural Networks (CNN) o Xarxes Neuronals Convolucionals en valencià. La gran rellevància d'este tipus de xarxes, degut a la seua aplicació en el reconeixement d'imatges o en el camp de la intel·ligència artificial, ha desembocat en un fort interès per la comunitat científica a desenrotllar nous mecanismes per a augmentar l'eficiència i el poder de còmput d'este tipus de xarxes neuronals. Una de les propostes que més força està cobrant és el desenrotllament de processadors sistòlics o de propòsit específic. Exemples d'això són la recent TPU de Google [9] o l'Eyeriss [10].

Este treball se centra en l'ús d'un simulador d'array o processador sistòlic cridat SCALE-Sim per al còmput de CNN per a identificar paràmetres crítics en l'arquitectura maquinari d'este tipus de dispositius per a posteriorment proposar canvis concrets en la seua configuració a fi de obtindre una major eficiència, maximitzant el ràtio entre productivitat, utilització i consum energètic.

Per a això primerament es fa un estudi detallat de l'arquitectura del propi simulador, identificant els seus paràmetres i comprenent el seu funcionament. El simulador llig uns fitxers de configuració per a la seua execució i genera uns arxius d'eixida amb els resultats sobre el comportament i rendiment que hauria tingut l'array configurat a l'executar una determinada xarxa neuronal. La comprensió dels fitxers d'entrada i sobretot els d'eixida és clau per a després analitzar els resultats i tractar d'optimitzar les configuracions.

Després de l'etapa de l'estudi del simulador amb què es treballa, es procedeix a l'etapa d'experimentació, triant un conjunt finit de CNN però variat procurant que siga el màxim representatiu possible i es procedeix a fer assajos canviant distints paràmetres en l'arquitectura del maquinari que s'està simulant per a estudiar l'impacte que tindria en les prestacions i consum.

Després d'esta etapa es procedeix a analitzar els resultats obtinguts i la derivació dels resultats que s'obtenen directament del SCALE-Sim en resultats addicionals com és el consum energètic. Per a això en l'estudi s'utilitzen les dades d'eixida del SCALE-Sim per a calcular les dades energètiques per mitjà del simulador CACTI.

Una vegada analitzats tots els resultats es proposen optimitzacions en les configuracions de l'arquitectura simulada.

Palabras clave: aceleradores para CNN, Salida Estacionaria, arrays sistólicos, Redes Neuronales Convolucionales, simulación, flujos de datos, consumo energético.

Keywords: CNN accelerators, Output Stationary, systolic arrays, Convolutional Neural Networks, simulation, data flows, power consumptions.

Tabla de contenidos

Tabla de Figuras

Índice de Tablas

1 CAPÍTULO: Introducción y objetivos

1.1 Introducción

Para ayudar a entender este proyecto primero se introducen conceptos básicos sobre los que se cimienta este proyecto. Una de las bases más importantes sobre las que se construye es el concepto de red neuronal. Una red neuronal es un conjunto de nodos, cada uno de ellos inspirado en el funcionamiento de una neurona cerebral biológica. Estos nodos, al igual que las neuronas cerebrales están interconectados entre sí de una forma específica mediante unas conexiones a través de las cuales se transmite la información entre neuronas. En una red neuronal cada neurona recibe información, que produce una respuesta en la neurona receptora y la transmite a otras neuronas a través de esas conexiones, al igual que pasa en nuestro cerebro, dónde pequeños impulsos eléctricos generados químicamente estimulan neuronas a través de una red de conexión. Como se puede observar son muchas las similitudes entre el funcionamiento de una red neuronal artificial y el cerebro humano en cuanto a estructura y funcionamiento y del mismo modo que el cerebro humano tiene la capacidad de aprender, las redes neuronales artificiales también la tienen, pero la explicación de cómo tiene lugar esto queda fuera del alcance de este trabajo.

En la siguiente imagen se puede apreciar de forma visual las similitudes entre una red neuronal artificial y el cerebro humano.

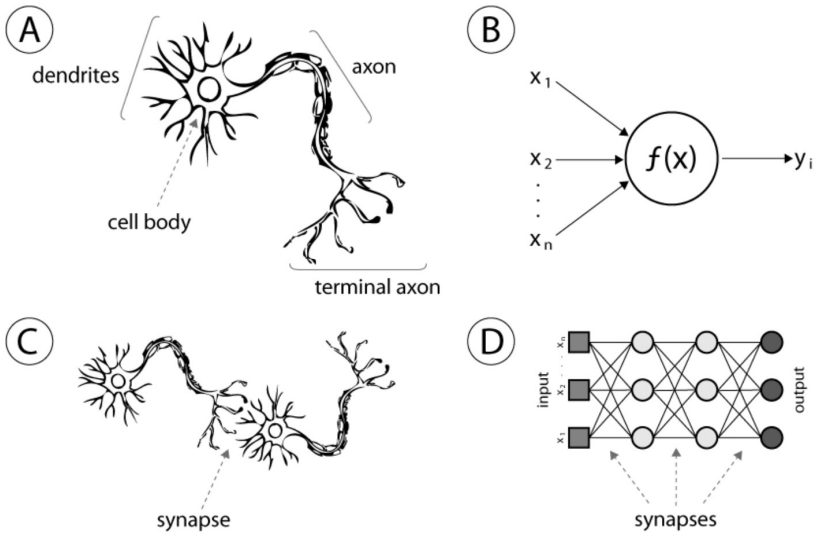

Figura 1:Comparación de elementos de una red neuronal biológica con una artificial [3]

Cuando se habla de redes neuronales, no se puede evitar la necesidad de darles un apellido, ya que estas se clasifican en varios tipos en función de su topología o su aprendizaje, como por ejemplo las redes neuronales recurrentes o las de estimulación frontal. A su vez hay subtipos, de modo que se puede encontrar una gran cantidad de apellidos para las redes neuronales.

Por ejemplo, este trabajo está centrado en un tipo de redes neuronales llamadas Redes Neuronales Convolucionales o Convolutional Neural Networks (CNN) en inglés, que son un tipo específico de una red neuronal conocida como Perceptrón Multicapa. Para profundizar un poco en cuanto a los tipos de redes neuronales y su clasificación en [5] se puede encontrar un artículo breve y sencillo.

Las redes neuronales convolucionales son un tipo de red neuronal que tiene especial similitud con las neuronas de la corteza visual primaria en el cerebro humano, y que resultan de especial interés para la clasificación, procesamiento y reconocimiento de imágenes en inteligencia artificial, algo que está muy al orden del día en el paradigma tecnológico actual y por tanto son un tipo de red neuronal muy importante hoy en día.

Las grandes demandas y la imparable evolución y desarrollo de las redes neuronales han hecho que cada vez sean más pesadas y complejas de computar para los procesadores de propósito general (como los que se encuentran en nuestros ordenadores de casa o en nuestro teléfono móvil). Sobre todo, porque éstos no se adaptar a la estructura de las mismas. Esto ha generado la necesidad de buscar optimizaciones y explorar nuevos caminos para adaptarse a la estructura y al crecimiento de las redes neuronales modernas.

Una de las alternativas propuestas y que más éxito está teniendo es el desarrollo, fabricación y uso de procesadores diseñados específicamente para computar redes neuronales. Estos procesadores se basan en arrays sistólicos y a diferencia de los procesadores de propósito general estos no han sido diseñados para realizar tareas de propósito general, sólo para el cómputo de redes neuronales. Es por ello por lo que ofrecen un rendimiento enormemente mejor que los procesadores de propósito general a la hora de computar estas redes, con el coste de no poder aprovecharse para desarrollar otras tareas como por ejemplo soportar la ejecución de un sistema operativo como Windows, Linux o Android que un procesador de propósito general sí está diseñado para hacer. De las características de los arrays sistólicos se habla más en detalle posteriormente en este documento.

La experimentación directa en un array sistólico real está al alcance de muy pocas personas, por ello en este proyecto se emplea un reciente simulador de uso libre llamado SCALE-Sim. Además, por tratarse de un simulador se tiene la flexibilidad de cambiar parámetros de la arquitectura. Este es un simulador de arrays sistólicos para redes neuronales convolucionales que

permite al usuario simular distintas configuraciones de la arquitectura de un array sistólico y simular ejecuciones (sobre esa arquitectura definida por el usuario) de redes neuronales convolucionales reales midiendo parámetros de rendimiento.

1.2 Motivación

Desde 1956 cuando se habló por primera vez en un congreso científico sobre redes neuronales, en Congreso de Dartmouth, este campo de la ciencia de la información no ha parado de evolucionar desde entonces hasta hoy en día, en la actualidad se podría decir que se está iniciando la época dorada de las redes neuronales y la inteligencia artificial.

En gran medida este gran avance en esta área se debe a la disponibilidad de dispositivos hardware cada vez más potentes y a la aparición de servicios en la nube que dan acceso a una enorme capacidad de cómputo sin tener que comprar supercomputadores, lo cual posibilita a muchas empresas desarrollar proyectos en I+D con una inversión mínima.

Lo más destacable de todo es que en las empresas que se dedican a ofrecer estos servicios en la nube como Google o empresas que se proveen de hardware para estos centros de datos como ARM, en los últimos años han identificado un gran volumen de demanda de computación de redes neuronales y programas de inteligencia artificial, lo cual las impulsa a investigar e invertir en métodos para aumentar el rendimiento a la hora de satisfacer este tipo de demanda cada vez más voluminosa.

Finalmente, el presente trabajo se ha centrado en el análisis de la arquitectura de los arrays sistólicos, ya que como se ha expuesto anteriormente, las redes neuronales juegan un papel fundamental en el futuro de la tecnología, y el desarrollo de estas depende en gran medida del desarrollo del hardware para soportarlas. Además, cabe añadir que este proyecto tenía la posibilidad de ser desarrollado y financiado por la conocida empresa HUAWEI. Esto último no hace más que evidenciar aún más el interés por parte de grandes empresas tecnológicas como esta por el desarrollo de un hardware cada vez más potente que soporte las cada vez más desarrolladas y demandantes redes neuronales del futuro.

1.3 Objetivos

En este proyecto se ha analizado la arquitectura de los arrays sistólicos por medio del simulador para aceleradores de CNN basados en un array sistólico modelado en el simulador SCALE-Sim. Para ello ha sido necesario aplicar conocimientos sobre Linux, redes neuronales, arquitecturas de procesadores y sistemas de memoria. La metodología utilizada ha consistido en ejecutar el simulador reiteradamente con distintas configuraciones y posteriormente analizar los resultados obtenidos y el impacto de los cambios en las configuraciones en los resultados.

Teniendo esto en mente, los objetivos que pretende este proyecto son:

1. La familiarización con el simulador SCALE-Sim y las cargas que soporta.
2. Configurar el array sistólico de acuerdo con el flujo de datos de Salida Estacionaria y obtener resultados de rendimiento.
3. Estudiar el impacto sobre el rendimiento de varios parámetros arquitecturales, principalmente relacionados con el tamaño del array y el subsistema de memoria.
4. Analizar los resultados obtenidos e identificar posibles optimizaciones en la configuración.

1.4 Impacto esperado

Se espera que este proyecto pueda ser de utilidad como punto de partida para aquellas empresas que estén buscando iniciar un proyecto de investigación más avanzado que el que se ha realizado aquí sobre arrays sistólicos, pudiendo tomar como resultados preliminares las conclusiones y los resultados contenidos en este trabajo.

Este documento puede ser de especial utilidad para aquellos usuarios que tengan interés por utilizar el simulador SCALE-Sim en sus proyectos o inicios a la investigación, o simplemente para cualquier entidad o persona interesada en conocer el impacto de ciertos parámetros arquitecturales de los arrays sistólicos o del subsistema de memoria para un array sistólico de CNN.

1.5 Metodología

La metodología seguida durante el desarrollo de este proyecto se puede dividir en diversos pasos:

1. Preparación de un entorno de trabajo y estructura de directorios en Linux con todos los elementos necesarios para cumplir los objetivos del proyecto.

2. Experimentación directa con el simulador. Primero, sin hacer cambios en las configuraciones y posteriormente modificando parámetros. Familiarización con el simulador CACTI.

3. Comprensión y análisis de los resultados obtenidos en el paso anterior con el objetivo de familiarizarse con el simulador SCALE-Sim.

4. Inspección del código y la arquitectura del software del simulador SCALE-Sim para entender todos los matices del funcionamiento y la forma en la que se procesan las entradas y el formato de la salida

5. Planificación de las ejecuciones que pueden ser interesantes y desarrollo de Script en Bash para automatizar el proceso.

6. Análisis de resultados directos, y desarrollo de un mecanismo para la obtención de resultados indirectos mediante script. Aplicación de los datos obtenido al simulador CACTI para la obtención de consumos energéticos.

7. Elaboración de las gráficas y extracción de conclusiones.

1.6 Estructura del documento

Este documento está estructurado de la siguiente manera:

- **Introducción y objetivos**: Contiene una breve introducción al contexto del trabajo, exponiendo la motivación y los objetivos que se desean alcanzar con el mismo. También se describe el impacto que se espera del mismo, la metodología empleada y las colaboraciones con otros proyectos.

- **Estudio estratégico**: Se describe el estado del arte, una breve reflexión sobre trabajos relacionados que se hayan desarrollado anteriormente en la escuela y el avance que supone la propuesta de este proyecto.

- **Análisis del problema**: Se hace un breve análisis más global del proyecto.

- **Conceptos técnicos**. Breve estudio sobre los conceptos que cimientan el proyecto.

- **Simulador utilizado**: Se detalla en profundidad las peculiaridades y el funcionamiento del simulador SCALE-Sim y los scripts desarrollados durante el proyecto.

- **Resultados experimentales**: Contiene resultados y análisis de los resultados obtenidos al hacer simulaciones variando distintos parámetros arquitecturales.

- **Conclusiones**: Contiene las conclusiones que se obtienen de todo el trabajo realizado.

- **Bibliografía**: Contiene la bibliografía empleada durante el proyecto.

1.7 Colaboraciones

Este **trabajo** es resultado de un **proyecto** de investigación llevado a cabo con los profesores que aparecen en la portada de este documento.

Este **proyecto** ha sido desarrollado conjuntamente y en paralelo con mi compañero Pau Castelló Ferrer quién ha elaborado el trabajo **"Prestaciones y energía de esquemas de mapeo sin precarga para aceleradores CNN sistólicos"**. En este trabajo se analiza y estudia el impacto en las prestaciones de diversos parámetros arquitectónicos en el array sistólico para CNN bajo el flujo de datos de Salida Estacionaria mientras que en el trabajo de mi compañero este estudio se realiza bajo los flujos de datos de Entrada Estacionaria y de Pesos Estacionarios.

La fase de análisis y experimentación del proyecto se ha hecho de forma conjunta y estrecha entre ambos autores por lo que estas fases en ambos trabajos presentan grandes similitudes.

A lo largo del documento se encontrarán referencias al otro trabajo y viceversa debido a que resulta interesante la comparación de ciertos resultados obtenidos en un trabajo con los obtenidos en el otro.

2 CAPÍTULO: Estudio estratégico

2.1 Estado del arte

En el contexto tecnológico actual existe un incesable incremento en el volumen de demanda de capacidad de cálculo para redes neuronales que ha provocado que compañías de gran renombre mundial que ofrecen servicios en la nube como Google empiecen a desarrollar aceleradores basados en arrays sistólicos para el cálculo de redes neuronales. Google ha desarrollado el suyo, bajo el nombre de TPU (Tensor Processing Unit) tal y como se puede comprobar e investigar de manera más detallada en [9]. Estos aceleradores han sido diseñados para instalarse en los centros de cálculo de Google para aumentar el poder de cálculo para redes neuronales.

Además, de esto, de forma natural también están apareciendo simuladores para modelar estos aceleradores como SCALE-Sim [11] a consecuencia del interés de las Universidades y la fuerte inversión en I+D de grandes empresas interesadas en desarrollar aceleradores cada vez más eficientes y potentes y añadirlos a sus centros de cálculo. Recientemente NVIDIA ha desarrollado también su acelerador para redes neuronales llamado NVDLA (NVIDIA's Deep Learning Accelerator) disponible en [13] y en dónde se encuentra el código necesario para fabricar los chips o para ser modificado por posibles interesados cómo ARM [12], ya que no es *open source*.

Queda expuesto pues el gran interés que existe actualmente por estos aceleradores y el hueco que se han ganado entre los objetivos de inversión para I+D. Pero el desarrollo de estos aceleradores no está limitado únicamente a gigantes como Google y NVIDIA, también se están desarrollando artículos científicos y presentaciones en congresos con nuevas propuestas en flujos de datos y parámetros de la arquitectura buscando la máxima eficiencia como en [10].

Si se buscan trabajos o proyectos acerca de arrays sistólicos, flujos de datos en los arrays sistólicos o aceleradores de CNN en la base de datos de proyectos realizados en la ETSINF de la UPV no se encuentran antecedentes a este proyecto. Por lo tanto, este proyecto y los dos Trabajos Fin de Grado que surgen del mismo rellenan un espacio en el conocimiento que hasta el momento estaba vacío en esta Escuela y aportan nuevo conocimiento, actualidad y originalidad.

3 CAPÍTULO: Análisis del problema

Este proyecto inspirado en las numerosas y actuales investigaciones que se están llevando a cabo sobre aceleradores para redes neuronales basados en arrays sistólicos pretende arrojar un poco más de luz acerca del impacto de parámetros arquitectónicos como el número de elementos de procesamiento o processing elements (PEs) que forman el array sistólico, el impacto de su dimensionamiento (altura y anchura del array de PE) y los tamaños de los buffers que alimentan dicho array empleando flujos de datos concretos. Para este trabajo se estudia la sinergia del flujo de datos basado en salida estacionaria con variaciones de los parámetros nombrados anteriormente

Para ello en este proyecto se ha empleado el simulador de aceleradores CNN basados en arrays sistólicos SCALE-Sim, ya que una de las ventajas que nos ofrece el simulador es que nos permite analizar el impacto que tienen los parámetros que se configuran, con independencia de limitaciones hardware. En otras palabras, el simulador no emplea parámetros como la frecuencia del bus o el ancho de banda máximo que se soporta al inyectar datos en el array. En vez de eso el simulador te indica los anchos de banda que serían requeridos para obtener el máximo rendimiento. No hay que limitarse a lo mencionado anteriormente sobre el ancho de banda ya que es sólo un ejemplo para esbozar la capacidad del simulador y su funcionamiento. Este discurso se realiza más en detalle en el *CAPÍTULO 5*.

El desarrollo de un proyecto como éste que requiera de experimentación de configuraciones sobre la arquitectura de arrays sistólicos es económicamente inviable, por tanto, se eligió emplear el simulador SCALE-Sim.

Además, del software del simulador también es necesario disponer de equipo informático con un sistema basado en LINUX. Dicho equipo informático debe suplir las grandes demandas de potencia de cálculo y capacidad de almacenamiento que requiere dicho simulador, pues este genera trazas con un gran volumen de espacio (del orden de varios Giga Bytes por ejecución). A esto hay que sumarle que cada ejecución oscila entre los 20 minutos y las 4 horas en un procesador Intel i5 de octava generación.

Si esto se escala a la gran cantidad de ejecuciones que son necesarias para realizar el estudio que se persigue se obtiene que se necesita aproximadamente más de 1TB de almacenamiento y varios días de ejecución. A todo esto, hay que añadirle que se hace casi imperativa la necesidad de automatizar el proceso de lanzamiento de ejecuciones empleando scripts, los cuales tienen que ser programados.

4 CAPÍTULO: Conceptos técnicos

Antes de introducir el funcionamiento del simulador Scale-SIM empleado en este proyecto es necesario repasar ciertos conceptos técnicos que forman el mayor peso de las bases de conocimientos que se requieren para entender el mismo.

4.1 Arrays sistólicos

Como ya se ha expuesto, estos aceleradores de redes neuronales basados en arrays sistólicos surgen como consecuencia directa de la necesidad de optimizar los cálculos sobre redes neuronales. Estos aceleradores están atrayendo enormes cantidades de inversión de capital privado de grandes empresas tecnológicas como ya se ha comentado, pero ¿por qué? ¿cómo de apreciable es la mejora que introducen estos aceleradores en el cálculo de redes neuronales? Como ya es sabido, una inversión si no es rentable se abandona y se busca la siguiente que pueda ofrecer rentabilidad. A continuación, se dará respuesta a las anteriores preguntas de la mano de *Google Cloud* y por qué ellos no abandonaron la inversión en investigación y desarrollo de su propio acelerador, la TPU de Google.

Observando el rendimiento que ofrece la TPU respecto a las CPU y las GPU en la computación de redes neuronales, los resultados son incontestables.

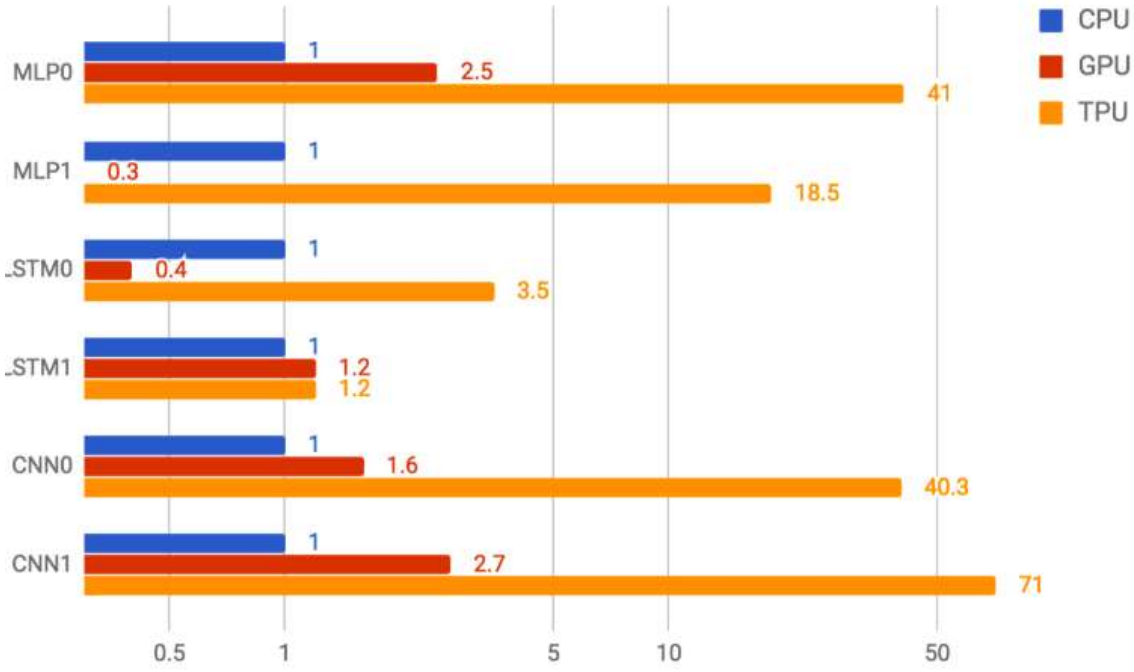

Figura 2:Comparación del rendimiento entre TPU, GPU y CPU [9]

Además de mejoras significativas en el rendimiento, los resultados no dejan de ser sorprendentes en el contexto de la eficiencia energética.

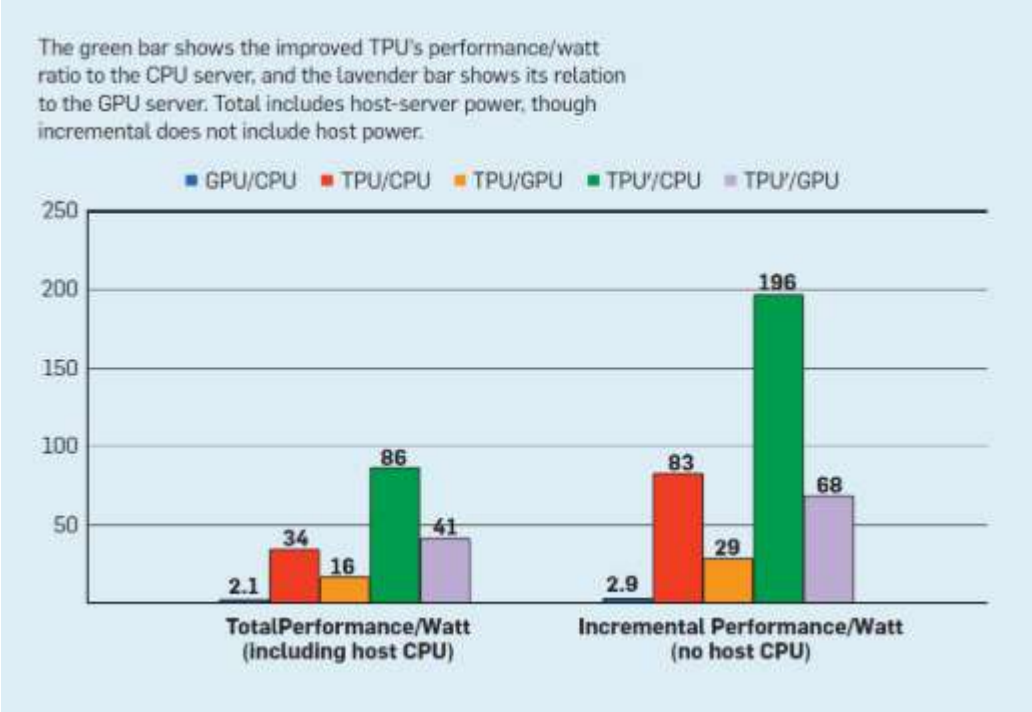

Figura 3: Comparación de la eficiencia energética entre TPU, GPU y CPU [9]

Viendo estos datos facilitados por Google Cloud, no cabe lugar a dudas de que la inversión fue rentable.

Antes de empezar con la arquitectura es interesante decir que la mayoría de los aceleradores, al igual que la TPU de Google, implementan un conjunto bastante modesto de instrucciones CISC (Complex Instruction Set Computer) a diferencia de muchos procesadores RISC (Reduced Instruction Set Computer) actuales.

Aunque existen muchos modelos de arquitectura para implementar un acelerador para DNN, este proyecto se centra en la arquitectura modelada en el simulador SCALE-Sim, a fin de dotar de concisión y claridad a este documento.

De este modo y a efectos prácticos para este proyecto se puede representar gráficamente la arquitectura de un acelerador basado en array sistólico y su integración en el sistema de la siguiente manera:

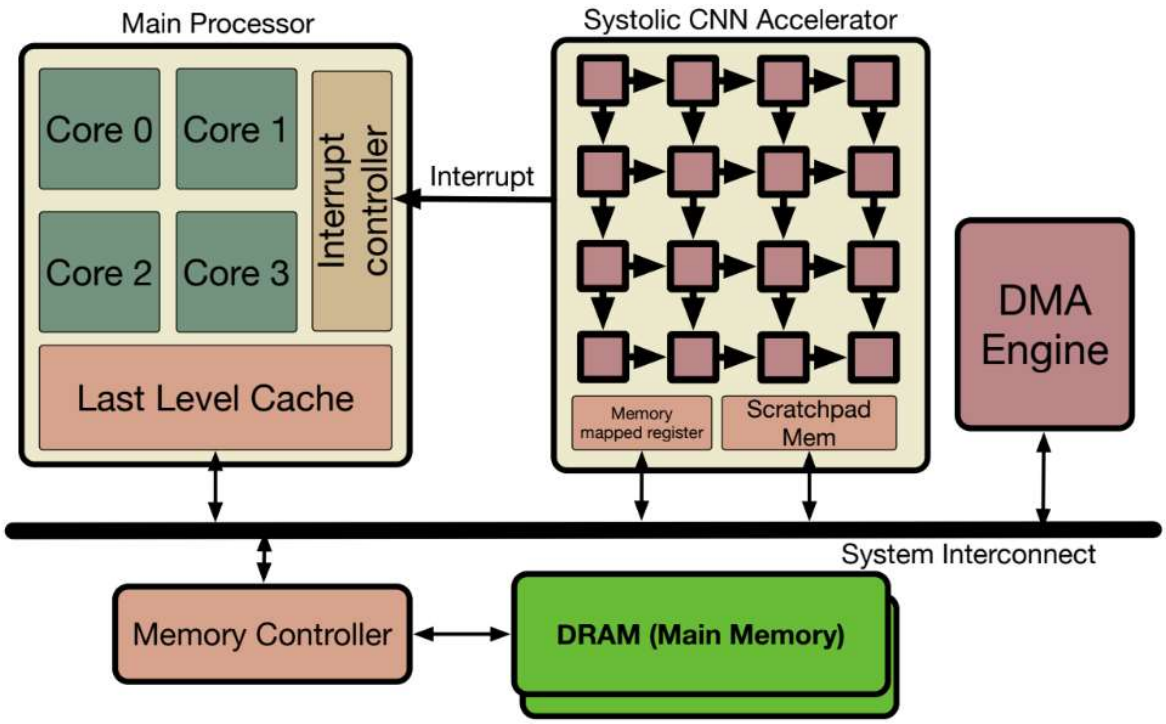

Figura 4: Acelerador para CNN sistólico y su integración en el sistema [11]

Centrándose en el análisis de los elementos que se observan dentro del chip del acelerador, éste se estructura en base a un sistema de memoria interno y un conjunto de unidades de procesamiento individuales llamadas **PE** (Processing Element) Estas unidades están representados en la figura por medio de cuadrados pequeños que ocupan la mayor parte del chip. Estos elementos de procesamiento están interconectados entre sí formando una malla. Tal y como se visualiza en la figura anterior, un PE puede recibir datos desde su izquierda y desde la parte superior y transmitir datos al elemento que ocupa una posición abajo en la malla y al de su derecha. Entender en qué sentido pueden circular los datos a través de la matriz de PE es crucial para entender el funcionamiento del array sistólico.

Estos elementos llevan a cabo dos operaciones únicamente, multiplicar y sumar ya que sólo son necesarias estas dos operaciones para la computación de redes neuronales tal y como se explica en este documento en la sección *4.2.1* de este mismo capítulo. La ejecución se realiza con un alto nivel de paralelismo directamente derivado del número de PE que componen el acelerador y de la carga que se esté ejecutando. A mayor número de PE mayor paralelismo se podría alcanzar en una ejecución siempre y cuando los datos lo permitan.

Pasando a los elementos de memoria, en general para los aceleradores de CNN se puede establecer una jerarquía de memoria multinivel.

El array sistólico modelado en SCALE-Sim está compuesto por una jerarquía de memoria de dos niveles ordenados de menor a mayor tiempo de acceso. El primer nivel está formado por la "***Scratchpad Memory***", la cual se puede identificar en la Figura del acelerador. Esta memoria está integrada en el propio chip del dispositivo y se trata de un triple buffer implementado sobre una memoria SRAM dividido en 3 particiones cada una con sus propios puertos de lectura y escritura. A efectos prácticos cada partición puede ser vista como un doble buffer independiente. La primera partición está reservada para el almacenamiento exclusivo de datos de IFMAP, la segunda para datos de FILTER y la tercera para datos de OFMAP. Estos conceptos tanto como las distintas formas en las pueden ser transmitidos los datos durante la ejecución dentro del array de PE (en base a las conexiones vistas anteriormente) se trata en la sección *4.2.2* de este mismo capítulo.

El segundo nivel está formado por la memoria principal del sistema en el que se ha integrado el acelerador y por lo tanto fuera del chip del acelerador, es decir la memoria DRAM, la cual tiene un tiempo de acceso muy elevado y por ello se trata de reducir lo máximo posible el número de accesos a esta memoria externa mediante la implementación de la memoria *Scratchpad* en el chip.

Esto es en el caso de la arquitectura en la que se basa el simulador SCALE-Sim tal y como se ha mencionado y es con la que se trabaja en este proyecto, pero algunas arquitecturas, como el Eyeriss están formados por una jerarquía de memoria de 3 niveles, introduciendo una memoria llamada GLB en el chip (Global Buffer) entre el nivel de *scratchpad* y el nivel de DRAM, a fin de reducir aún más el número de accesos a la memoria externa DRAM.

El acelerador ocupa un puesto de esclavo en el sistema en el que se instale, quedando sometido a las instrucciones de un maestro (generalmente un procesador), formando un sistema maestro-esclavo.

El procesador envía las órdenes de ejecutar las tareas de DNN que necesite junto con los datos que vaya a necesitar para su ejecución en el acelerador (como por ejemplo las direcciones de DRAM donde leer los elementos que necesita computar) y mientras el procesador puede seguir ejecutando otras tareas a la vez que el acelerador realiza los cálculos que le han sido ordenados, cuando los cálculos finalizan se acaban de escribir todos los datos en la memoria DRAM y se envía una interrupción al procesador para avisar de que los datos están listos y el procesador ya decide cómo y cuándo manejar esos datos y si le envía más tareas al acelerador o no.

La descripción de este funcionamiento enlaza con la explicación del último elemento por analizar en el chip del acelerador, que es la memoria "***Memory mapped register***". La función de esta memoria es simplemente la de almacenar los descriptores de tarea que el maestro le envía al acelerador.

4.2 Flujos de datos

El segundo concepto técnico más importante y que se podría decir que es el principal objeto de estudio de este proyecto sería el de flujo de datos.

4.2.1 Introducción a los flujos de datos

Para que una unidad de procesamiento de información sea capaz de realizar operaciones con unos datos de entrada para generar datos de salida es necesario que los datos que se van a utilizar durante el cálculo estén físicamente en el dispositivo. Estos datos suelen estar generalmente almacenados en pequeñas, pero extremadamente rápidas memorias como son los registros en el caso de los procesadores, por ejemplo. Además, también pueden existir distintos niveles de jerarquía de memoria dentro del acelerador como es el caso del modelo implementado en el SCALE-Sim tal y como se explicó en la sección anterior referente a la arquitectura de arrays sistólicos.

Una vez queda claro que los datos deben estar disponibles físicamente en el dispositivo de procesamiento, la siguiente pregunta natural que se podría hacer es cómo se gestiona el almacenamiento y el movimiento de esos datos dentro del dispositivo. La respuesta a esta pregunta será que depende de la definición del flujo de datos que se utilice en el array sistólico.

De este modo se puede entender el concepto de **flujo de datos** como la política o mapeo que siguen los datos dentro del dispositivo. Es por ello por lo que en algunas ocasiones se puede referir a este concepto tanto como por el nombre de flujo de datos (o Data Flow en inglés) como por el nombre de mapeo de datos, ya que igual que en un mapa, se permite ver la ruta que siguen los datos dentro de la unidad de procesamiento.

Esta política que define el movimiento de datos no sólo atañe a los datos de entrada con los que se va a obtener una salida, sino también a los datos que se obtienen durante el cómputo.

Las posibilidades de mapeo de datos dependen tanto de la arquitectura del dispositivo como del ingenio de los diseñadores a la hora de establecer esas políticas para el mismo. Y aunque los

tipos de flujos de datos que sean aplicables a una unidad de procesamiento están definidos y limitados por su propia arquitectura, en muchas ocasiones es el segundo factor(el ingenio) el que en base a las reglas de la arquitectura es capaz de idear y crear distintas políticas para un mismo hardware, pudiendo dar lugar a algunas opciones mejores que otras en todos los aspectos aumentando el rendimiento del hardware, o sólo en algunos aspectos concretos permitiendo optimizaciones en el rendimiento del hardware para contextos específicos.

Es por esto por lo que el flujo de datos es tan importante en una unidad de procesamiento y en concreto en los arrays sistólicos, ya que sin necesidad de hacer cambios en el hardware o apenas añadiendo algunos cambios es posible aumentar su rendimiento considerablemente si se desarrollan nuevos flujos de datos o si simplemente se emplean los óptimos para el problema a resolver.

En este proyecto se analizaron los diferentes flujos de datos que se emplean en el simulador Scale-SIM y aunque este trabajo esté centrado en los mapeos sin precarga, el otro trabajo derivado del proyecto, ya mencionado en la sección *1.7 colaboraciones del CAPÍTULO 1* puede complementarse con éste para que el lector pueda tener una visión total sobre los flujos de datos y complementar los conocimientos que aquí se exponen.

4.2.2 Computación de CNN y tipos de flujos de datos

Una vez expuesto el concepto de flujos de datos y su importancia, el siguiente paso es exponer el funcionamiento general de una red CNN y las variaciones a la hora de implementar los cálculos en el hardware, dando lugar a diferentes tipos de flujos de datos.

Para hablar del cómputo de una red CNN, se habla del cómputo de una red neuronal que como tal su representación matemática es construida sobre 3 matrices. La primera de ellas es la matriz de entrada (**IFMAP**), en la cual se almacenan todos los datos de entrada que se desean inyectar en la red neuronal. La segunda matriz es la matriz de pesos o filtros (**WEIGHT** o **FILTER**), que contiene los datos sobre el conocimiento que tiene en ese momento la red neuronal. Finalmente, la tercera matriz es la matriz de salida (**OFMAP**), que contiene los datos que genera la red CNN de computar la matriz de entrada con la matriz de pesos- Esta idea es necesaria para entender el concepto de precarga explicado posteriormente en esta sección.

La forma de computar una red neuronal algorítmicamente es compleja y poco intuitiva pero necesaria para la total comprensión del proyecto. De este modo y en base a los conocimientos

expuestos en Eyeriss [10] se puede representar gráfica y matemáticamente el cómputo de una red neuronal por capas de la siguiente manera:

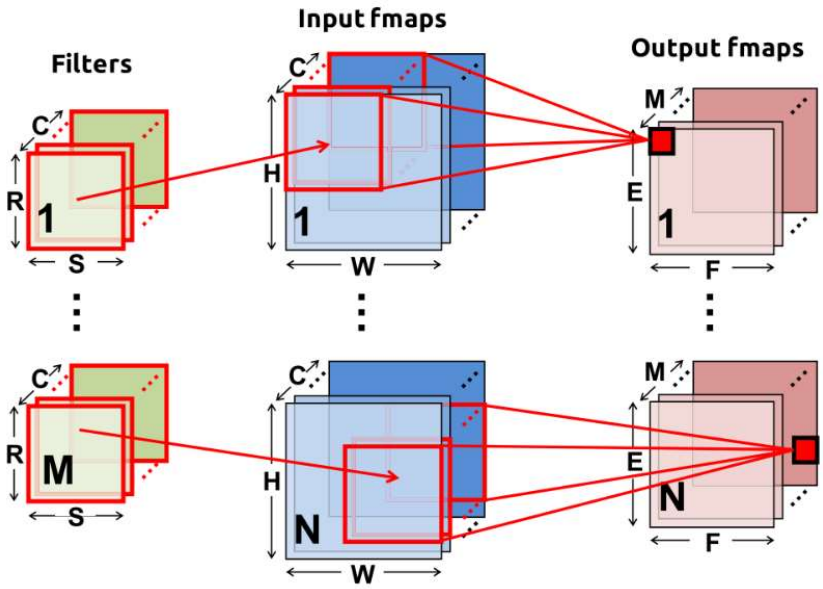

Figura 5: Representación gráfica del cálculo de una capa de CNN [10]

$$\mathbf{O}[z][u][x][y] = \mathrm{ReLU}\left(\mathbf{B}[u] + \sum_{k=0}^{C-1}\sum_{i=0}^{R-1}\sum_{j=0}^{S-1}\mathbf{I}[z][k][Ux+i][Uy+j] \times \mathbf{W}[u][k][i][j]\right),$$

En donde **O** representa la matriz de salida, **I** representa la matriz de entrada, **W** es la matriz de pesos o filtros y **B** es la matriz de sesgos. Es posible que el lector se pregunte sobre esta última matriz ya que no se ha mencionado anteriormente de su existencia en este documento debido a que para el alcance de este proyecto carece de relevancia, aunque se puede explicar cómo una matriz necesaria para el cómputo de la red neuronal, pero que no forma parte de la propia red (la cual está formada por las tres matrices anteriormente nombradas). La función de esta matriz es modificar la función que decide si se activan o no las neuronas que conforman la red añadiendo una constante como un valor de entrada más a la neurona, con el objetivo de que ese valor consiga ajustar el umbral de activación de las neuronas al valor deseado. Esta matriz es análoga a la función ReLU.

La función ReLU es una función matemática cuya finalidad es eliminar la linealidad en el resultado salida de la neurona. Eliminar la linealidad se puede entender como un proceso por el cual la neurona sólo puede generar dos resultados de salida posibles, activación (generalmente 1) o no activación (generalmente 0), evitando valores intermedios como por ejemplo 0,7263.

La función ReLU es la más empleada en CNN debido a que no se satura si el valor de salida es muy alto, tal y como sucede con otras funciones como por ejemplo la sigmoide. Pero la ReLU no es la, única que se emplea en redes neuronales, hay muchas más como la mencionada anteriormente (la sigmoide), la función hipertangente, la función identidad y muchas más. De hecho, existen incluso variaciones de cada una de estas funciones como por ejemplo la función sigmoide bipolar o la función sigmoide binaria. De este modo se evidencian las muchas posibilidades que existen a la hora de aplicar una función en este lugar para una red neuronal.

A continuación se exponen gráficamente las funciones nombradas anteriormente.

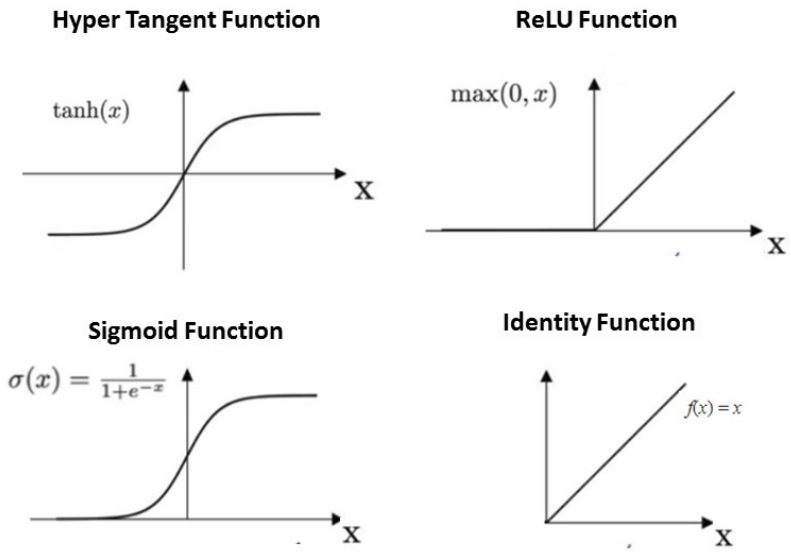

Figura 6: Algunas funciones de activación comúmnente usadas [7]

Sin embargo, todas ellas cumplen la misma función, eliminar la linealidad en la salida de la neurona. En redes neuronales las funciones encargadas de esto reciben el nombre de **funciones de activación**.

Retomando la fórmula anterior, es necesario saber que,

$$0 \leq z < N, \quad 0 \leq u < M, \; 0 \leq y < E, \; 0 \leq x < F$$

$$E = (H - R + U)/U, \quad F = (W - S + U)/U$$

Los cálculos de los valores **E** y **F** se deducen del álgebra matricial para la multiplicación de matrices, teniendo en cuenta que, **U** es un parámetro llamado tamaño del paso o stride size en inglés, al cual se le dotará de sentido a continuación en el documento.

Antes de seguir con la línea argumental del documento es necesario aclarar que los elementos(datos) individuales que conforman las matrices de la red neuronal se denominan **píxeles**.

A continuación, centrando la atención en la *Figura 2* se puede exponer que la forma en la que se calcula cada uno de los píxeles forman la matriz de salida son el resultado de multiplicar un conjunto píxeles de la matriz de pesos por un conjunto de píxeles de la matriz de entrada, esta matriz da como resultado una nueva matriz, a la cual se le aplica un proceso de reducción (suma) dando como resultado final un único pixel de la matriz de salida.

Es necesario hacer notar que generalmente se habla de matrices tridimensionales cuándo se hace referencia a las mismas, tal y como se ilustra en la representación gráfica, y que a su vez se tiene un conjunto de matrices tridimensionales, resultando la composición de las matrices de 4 dimensiones, siendo este último valor (el número de matrices 3D) el valor de la cuarta dimensión.

Generalmente la matriz de entrada será mayor a la de salida y por lo tanto el proceso descrito anteriormente de multiplicar y reducir submatrices es habitual que se realice varias veces con submatrices distintas durante la interacción entre ambas, pues es necesario computar todos los elementos de la matriz de filtros con todos los elementos de la matriz de entrada. Esto se realiza mediante un desplazamiento de los elementos que conforman la submatriz de entrada, este desplazamiento recibe el nombre de **tamaño de paso o stride size** y se aplica reiteradamente hasta que se ha recorrido la matriz, para todas las matrices de entrada.

Otro aspecto importante es que el tamaño de la submatriz de entrada es igual al tamaño de la matriz de pesos por motivos obvios, ya que el objetivo es que la matriz de filtros se desplace a lo largo de todo el ancho y la altura del conjunto total de datos de entrada, recogiendo en la matriz de salida las respuestas de los filtros ante las entradas.

Tras esta explicación apoyada por la representación gráfica del proceso de cálculo se puede retomar la representación matemática leyéndola fácil, resumida y claramente cómo sigue: Cada píxel de la matriz de salida es el resultado de aplicar la función de activación al valor de sesgo asociado más la multiplicación y reducción de la matriz de filtros con la submatriz de entrada asociada. Con asociada se hace referencia a que son los elementos necesarios que se requieren para calcular el píxel de salida que se quiere calcular. El proceso se repite hasta haber calculado todos los píxeles de salida dando como resultado el fin del cálculo de la capa de CNN.

Para que el lector adquiera la visión completa sobre el cómputo de CNN que necesita, es pertinente visualizar gráficamente el cómputo a nivel de neurona.

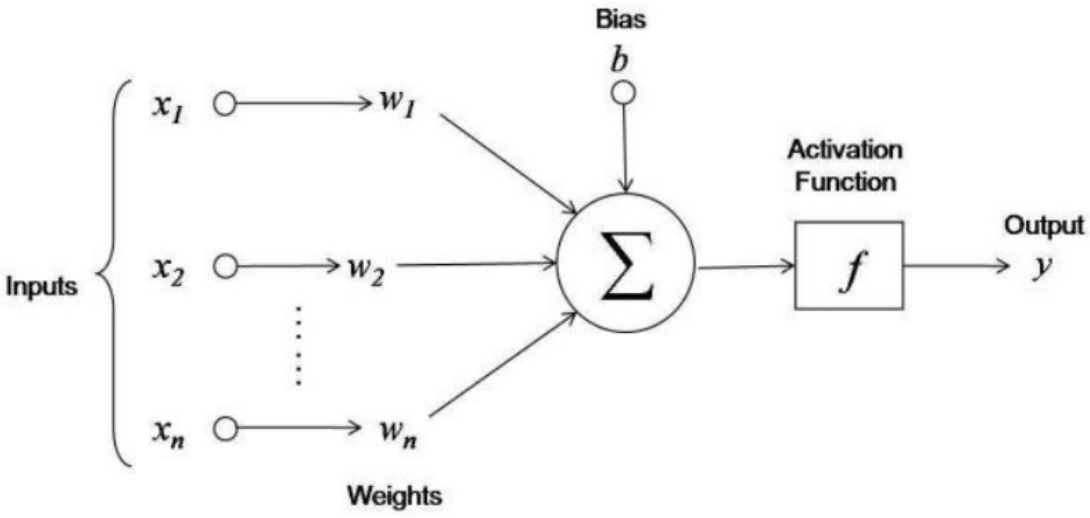

Figura 7: : Representación desde un punto de vista computacional de una neurona [8]

En esta Figura se puede apreciar de forma clara y distinta la alta correspondencia a nivel de neurona con el algoritmo de cálculo de una capa entera.

Es necesario recordar que el algoritmo expuesto anteriormente se utiliza para el cálculo de una única capa y que frecuentemente las redes CNN son multicapa, eso significa que el resultado que se obtiene del cómputo de la red neuronal es el resultado de computar todas y cada una de sus capas.

Una vez entendidos los cálculos que se realizan para el cómputo de una red CNN se puede entender cómo los flujos de datos utilizados se pueden dividir en dos grupos en función a una característica concreta llamada **precarga**. La precarga consiste en diseñar el flujo de datos alrededor de la idea de cargar en el array un conjunto de datos antes de comenzar la ejecución. La idea se basa en escoger una de las dos matrices de entrada que se ha explicado anteriormente y dividirla en conjuntos de datos con un tamaño suficiente como para poder ser almacenados todos en el dispositivo, dando lugar a una ejecución segmentada, donde se computan los elementos cargados previamente a la ejecución con los elementos de la otra matriz que serán inyectados secuencialmente obteniendo datos de salida, una vez todos los cálculos en los que eran requeridos los datos precargados han sido realizados, se procede a vaciarlos, a guardar los outputs generados y a precargar el siguiente conjunto de datos que serán computados en la siguiente paso de ejecución.

De este modo los tres flujos de datos empleados en el simulador pueden ser clasificados en **flujos con precarga** y **flujos sin precarga**.

5 CAPÍTULO: Simulador utilizado

SCALE-Sim, como se ha introducido en capítulos anteriores, es un simulador para aceleradores de CNN basados en arrays sistólicos. Este simulador permite modelar y replicar el comportamiento de un acelerador en base a unos parámetros de configuración de su arquitectura, tal y como se describe en la sección posterior. Una vez se configura la arquitectura se pueden ejecutar cargas reales de redes DNN y obtener resultados sobre la ejecución de esa carga que habría tenido lugar en la arquitectura configurada. El desarrollo de este proyecto se centra en la explotación de este funcionamiento para experimentar con los parámetros arquitecturales ejecutando distintas redes neuronales y comparando resultados entre las distintas configuraciones para las mismas cargas como se observará en el *Capítulo 6*.

Uno de los puntos débiles de este simulador es que no contempla parámetros como el ancho de banda disponible para acceder a la memoria del sistema o la frecuencia del propio acelerador. El simulador obvia estos parámetros para centrarse únicamente en el impacto en prestaciones y energía de los parámetros que modelan la arquitectura del propio acelerador simulado, dejando a un lado interferencias en los resultados que podrían ser causadas por limitaciones hardware. Esto puede afectar a los resultados obtenidos, siendo mejores de lo que se obtendrían en el sistema real con las limitaciones hardware reales Por lo tanto es una característica del simulador utilizado que se debe tener en cuenta en todo momento a la hora de trabajar con el simulador, sus resultados de salida y a la hora del tratamiento e interpretación de los datos.

Por otra parte, una de las mayores ventajas de SCALE-Sim es que nos permite simular el comportamiento y la interacción entre el acelerador y el sistema de memoria en el cual se integra, proporcionando una simulación muy precisa con trazas de los accesos tanto a las memorias de *scratchpad* como a la memoria fuera del chip DRAM.

Una de las características más importantes que deben ser expuestas sobre el simulador es la relativa a los flujos de datos que soporta. SCALE-Sim soporta tres tipos de flujos de datos:

- Entrada estacionaria: La entrada estacionaria o **IS** (*Input Stacionary* en inglés) es un tipo de flujo de datos con precarga que consiste en precargar reiteradamente subconjuntos de la matriz de IFMAP e ir inyectando los píxeles de la matriz de FILTER que necesitan para obtener píxeles de OFMAP. El proceso se repite mientras queden píxeles de IFMAP por computar.

- Pesos estacionarios: Los pesos estacionaria o **WS** (*Weight Stacionary* en inglés) es un tipo de flujo de datos con precarga que consiste en precargar reiteradamente subconjuntos de la matriz de FILTER e ir inyectando los píxeles de la matriz de IFMAP que necesitan para obtener píxeles de OFMAP. El proceso se repite mientras queden píxeles de FILTER por computar.

- Salida estacionaria: La salida estacionaria u **OS** (*Output Stacionary* en inglés) es un tipo de flujo de datos sin precarga que consiste en inyectar a cada PE los píxeles de IFMAP y FILTER necesarios para obtener un píxel de OFMAP, cuando el pixel de OFMAP está calculado éste se almacena en memoria y se procede a calcular el siguiente.

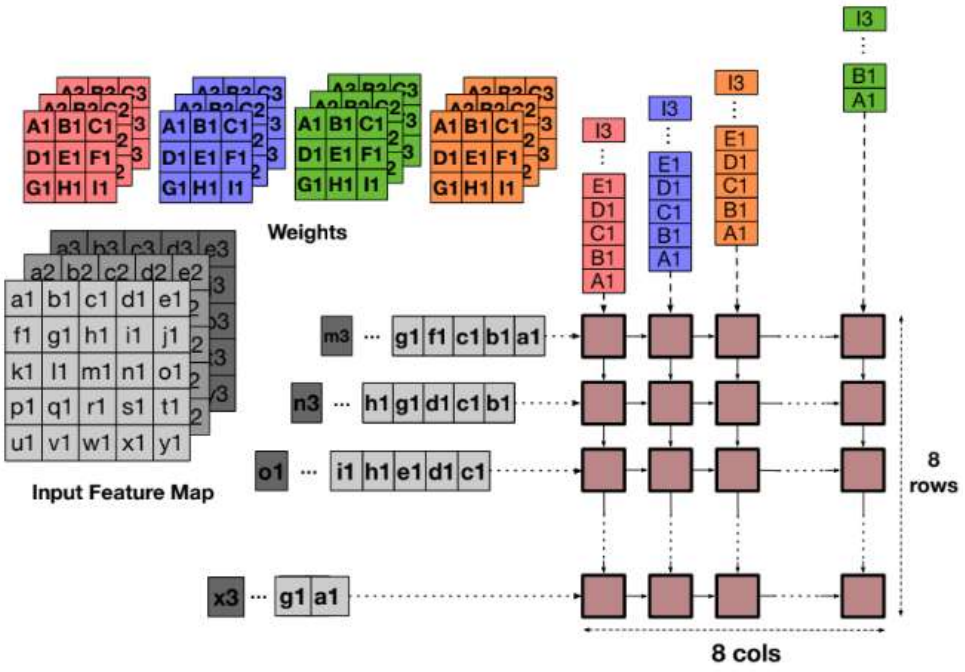

Figura 8: Mapeo de datos en el array con salida estacionaria [11]

En esta figura del documento del SCALE-Sim se aprecia gráficamente el mapeo de acuerdo con lo explicado anteriormente. En el primer ciclo en el primer elemento que ocupa la esquina superior izquierda se inyecta el primer pixel de FILTER por la parte superior y el primer pixel de IFMAP por el lateral izquierdo. El PE realiza los cálculos necesarios con esos píxeles y en el siguiente ciclo transfiere el píxel de FILTER al elemento inferior y el pixel de IFMAP al elemento situado en su lateral derecho. Este comportamiento se repite en todos los PE durante la ejecución del array asegurando una gran reutilización de los datos dentro del array. Gracias a esto, durante

la elaboración del proyecto se ha visto cómo en el caso de que no haya ciclos de parada durante la ejecución (algo que siempre se cumple en el simulador empleado en este proyecto) y de que la carga cuya ejecución se va a simular sea lo suficientemente grande como para llenar los elementos del array, se puede precalcular a priori el ciclo en el que el array estará a plena utilización de la siguiente forma: Altura del array + Anchura del array – 1

En este ciclo el último elemento del array situado en la esquina inferior derecha recibe los primeros píxeles de entrada y comienza a computar.

En la práctica el simulador realiza la primera escritura en el buffer de OFMAP en el momento en el que una columna del array tiene un píxel de OFMAP calculado(esta siempre será la primera columna del lateral izquierdo), de modo que se escribe toda la columna, un ejemplo real sobre esto se encuentra en la *sección 5.3* de este capítulo en la *Figura 19* donde se ve la traza de escritura en el buffer de OFMAP de un acelerador con matriz de PE con dimensiones 8x8 en donde en cada ciclo se escriben 8 píxeles de OFMAP correspondientes a una columna de 8 PE.

Como ya se había anunciado, este documento se centra en el estudio de los flujos sin precarga que se emplean en Scale-SIM, esto es el flujo de datos de entrada estacionaria u OS. Para obtener información más detallada y precisa acerca de los dos flujos de datos con precarga que soporta SCALE-Sim se recomienda leer el trabajo colaborador a este proyecto.

5.1 Arquitectura base

En base a los conocimientos expuestos en el capítulo 4 sobre arrays sistólicos, a continuación, se expone cómo el simulador modela los principales parámetros de la arquitectura del acelerador.

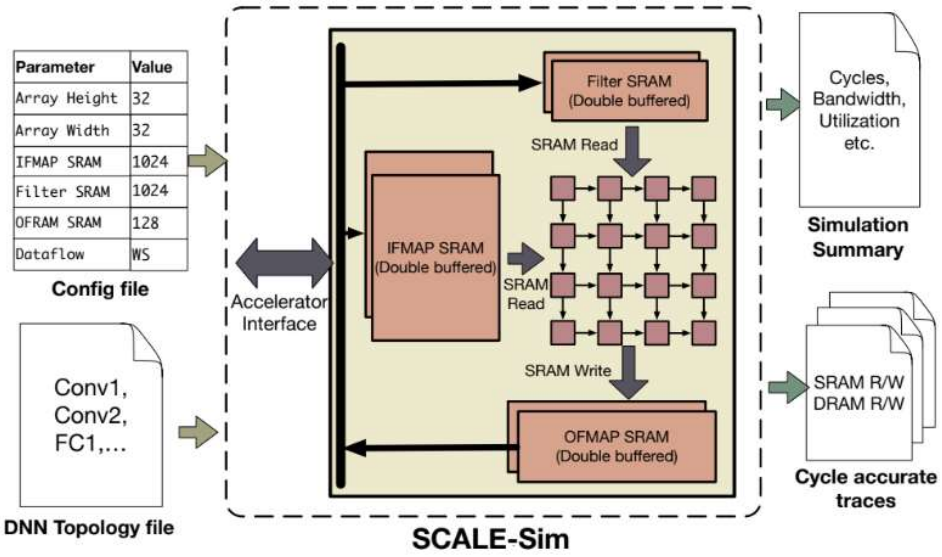

Figura 9: Configuración y parámetros del simulador SCALE-Sim [11]

En esta otra figura del documento de SCALE-Sim se aprecia más en detalle la organización interna de la memoria del propio acelerador, esta vez pudiéndose ver las tres particiones del triple buffer SRAM siendo una la de IFMAP, otra la de FILTER y finalmente la de OFMAP de las cuales ya se ha hablado. SCALE-Sim configura la arquitectura del acelerador a partir de un fichero de configuración con extensión ".cfg" en el que se concretan los parámetros relativos a la altura del array de PE (Array Height), su anchura (Array Width), los tamaños de los 3 buffers en KB (IFMAP SRAM, Filter SRAM y OFRAM SRAM) y por último el flujo de datos (Dataflow). Estos son los parámetros referentes a la arquitectura hardware del acelerador que modela el simulador, aunque a la hora de configurar el archivo la apariencia es la siguiente:

```
[general]
run_name = "MLPERF_AlphaGoZero_32x32_os"

[architecture_presets]
ArrayHeight:    32
ArrayWidth:     32
IfmapSramSz:    512
FilterSramSz:   512
OfmapSramSz:    256
IfmapOffset:    0
FilterOffset:   10000000
OfmapOffset:    20000000
Dataflow:       os
```

Figura 10: Archivo de configuración scale.cfg [11]

Algunos nombres cambian respecto a la Figura 9 cómo por ejemplo los identificadores de los tres buffers, pero a pesar de ello resultan fácilmente reconocibles. También existen otros parámetros en el archivo como "*run_name*" que es el nombre con el que el simulador genera al directorio en el cual almacenar los resultados (es recomendable emplear nombres representativos). Los otros tres parámetros nuevos son las direcciones base a partir de las cuales el simulador genera las direcciones que aparecen en las trazas de accesos a memoria en el directorio de resultados correspondiente. Cada buffer tiene una dirección base que establece el usuario de forma que después puede ser fácilmente identificable en las trazas qué tipos de datos se están leyendo o escribiendo y en qué memoria.

En base a esta construcción el simulador también recibe una red neuronal para simular la ejecución de esa red en el acelerador definido en el archivo de configuración.

5.2 Cargas soportadas

Las cargas mencionadas en la sección anterior cuya ejecución se quiere simular en la arquitectura deben cumplir una serie de requisitos. En primer lugar, deben ser redes CNN o en su defecto redes neuronales fully connected. Esto último quiere decir que la red está compuesta por capas que están totalmente conectadas formando una función $\mathbb{R}_m \rightarrow \mathbb{R}_n$ o en otras palabras cada neurona recibe un input de todas las neuronas que forman la capa anterior.

En segundo lugar, una vez la red que se desea ejecutar en la simulación cumple el primer requisito, esta red debe representarse de acuerdo con el formato requerido por el simulador para los archivos ".csv":

Layer name	IFMAP Height	IFMAP Width	Filter Height	Filter Width	Channels	Num Filter	Strides
Conv1	416	416	3	3	3	16	1
Conv2	207	207	3	3	16	32	1
Conv3	103	103	3	3	32	64	1
Conv4	52	52	3	3	64	128	1
Conv5	26	26	3	3	128	256	1
Conv6	13	13	3	3	256	512	1
Conv7	11	11	3	3	512	1024	1
Conv8	9	9	3	3	1024	1024	1
Conv9	7	7	1	1	1024	125	1

Figura 11: Formato de los archivos .csv para las cargas [14]

En estos archivos se escribe una entrada por cada capa de la red indicando el nombre de ésta, a continuación, la altura, y la anchura de la matriz de IFMAP de esa capa seguidas de la altura y la anchura de la matriz de FILTER de la misma capa y después el número de canales de la matriz de IFMAP y FILTER (número de matrices de filtros) y para terminar el tamaño de paso. Hay que recordar que todos estos parámetros fueron explicados en la *sección 4.2.2* del *Capítulo 4.*

Todos los campos deben estar separados por una coma, lo cual lo hace compatible con los formatos de hoja de cálculo.

Aunque el simulador contiene por defecto numerosas cargas ya representadas con este formato, el usuario puede definir sus propias cargas.

A pesar de ello, en este proyecto se han escogido las mismas cargas que se estudian en [11] para poder comparar nuestros resultados con los del artículo, y de esta forma validar la corrección de los resultados obtenidos de las ejecuciones, así pues, las cargas empleadas en este proyecto son las siguientes y se representan de la siguiente manera:

Tag	Description
W1	AlphaGoZero [18]
W2	DeepSpeech2 [19]
W3	FasterRCNN [20]
W4	Neural Collaborative Filtering [21]
W5	Resnet50 [17]
W6	Sentimental CNN [22]
W7	Transformer [23]

Tabla 1:Cargas utilizadas en el proyecto [11]

Esta tabla es de gran importancia para este trabajo ya que a lo largo de este documento se hace referencia a las cargas sólo por su etiqueta o *tag*.

Todas estas cargas, como ya hemos dicho, vienen incluidas junto con el simulador, pero la carga W6 contiene un error en el archivo de configuración que provoca errores en la ejecución tal y como se puede ver en esta captura:

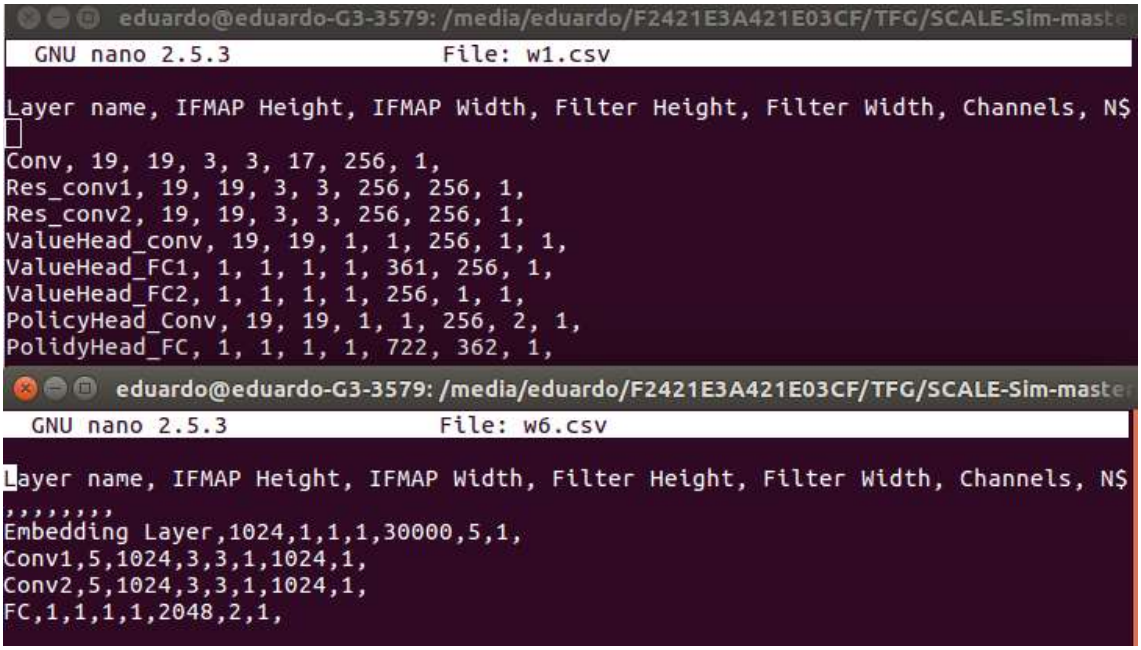

Figura 12: Captura ilustrativa de errores en la carga w6

El archivo abierto arriba corresponde al archivo w1.csv y el inferior corresponde a w6 ambos sin recibir modificaciones tras la descarga del simulador. Tal y cómo se puede apreciar en w6

existe una línea de comas que el simulador interpreta como nulos. Para poder ejecutar las cargas esta línea ha de ser eliminada. Para poder hacerlo se ha de abrir el archivo con un editor de texto plano, de lo contrario al abrirse con cualquier hoja de cálculo las comas no serán visibles y aparentemente todo parecerá estar correcto.

Además de esto la capa "*Embedding Layer*" no sigue el mismo patrón que todas las demás capas de las demás cargas siguen de no tener espacios y en su lugar todas tienen una barra baja "_". Esto último no produce error en el simulador, pero a la hora de ejecutar los scripts desarrollados en este proyecto supone un problema. Por tanto, se recomienda sustituir el espacio por una barra baja adoptando la misma forma que todas las demás capas del resto de cargas, quedando de esta manera "*Embedding_Layer*". Para entender concretamente el problema que esto ocasionaba en el script encargado de tratar los datos resultantes de las ejecuciones si resulta de interés se puede analizar el script en el *ANEXO 2*.

5.3 Datos de salida

Como se puede ver en la *Figura 9* de la *sección 5.1* en este mismo capítulo, el simulador a parte de los archivos de entrada que ya han sido comentados genera como salida una serie de archivos en los que se registra información acerca de la simulación realizada. Estos resultados vienen divididos en diversos ficheros. Tras ejecutar una carga el simulador genera una carpeta llamada "outputs" en la cual se almacenan todas las ejecuciones, dentro se encontrarán carpetas con el nombre de cada ejecución, dicho nombre definido en base a uno de los parámetros de configuración ya descritos A continuación y hasta el final de esta sección las capturas de pantalla que se van a mostrar son todas pertenecientes a la ejecución de la carga W1 de acuerdo con la *Figura 12* ejecutada sobre un acelerador con dimensiones 8x8 con flujo OS. De este modo el aspecto de un directorio de salida es el siguiente:

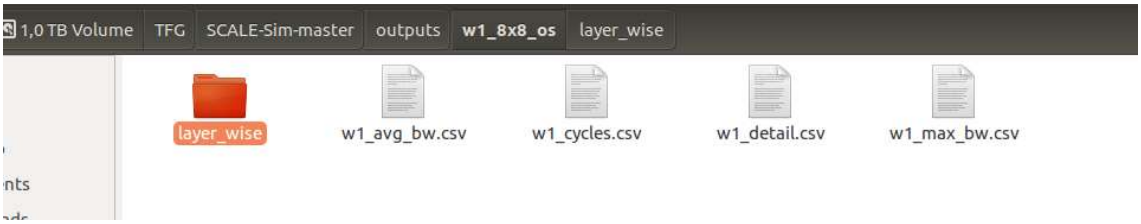

Figura 13: Ejemplo real de un directorio de salida

En el fichero "_max_bw.csv" se encuentran los anchos de banda máximos que se requerirían para alcanzar una ejecución sin ciclos de parada ocasionados por memoria. De este modo el archivo contiene los tamaños de los tres buffers SRAM, el ancho de banda máximo que alcanzaría la ejecución para lectura de DRAM de datos de IFMAP, FILTER y escritura de OFMAP. También se aportan el máximo ancho de banda que se registra en lectura de los buffers y también de escritura.

De este modo, aunque anteriormente se ha hablado acerca de que el simulador no tiene como parámetro configurable el ancho de banda, este ofrece el ancho de banda que se requeriría para soportar la ejecución sin necesidad de hacer ciclos de parada.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
IFMAP SRAM Size		Filter SRAM Size		OFMAP SRAM Size		Conv Layer Num		Max DRAM IFMAP Read BW		Max DRAM Filter Read BW		Max DRAM OFMAP Write BW		Max SRAM Read BW		Max SRAM OFMAP Write BW
524288		524288		262144		Conv		10		10		10		16		8
524288		524288		262144		Res_conv1		10		10		10		16		8
524288		524288		262144		Res_conv2		10		10		10		16		8
524288		524288		262144		ValueHead_conv		10		10		10		16		1
524288		524288		262144		ValueHead_FC1		10		10		10		16		8
524288		524288		262144		ValueHead_FC2		10		10		1		16		1
524288		524288		262144		PolicyHead_Conv		10		10		10		16		2
524288		524288		262144		PolicyHead_FC		10		10		10		16		8

Figura 14: Ejemplo real sobre un archivo "_max_bw.csv"

" _avg_bw.csv" contiene la misma estructura que el fichero "_max_bw.csv" pero con el ancho de banda medio registrado durante la ejecución simulada en lugar del ancho de banda máximo.

En "_detail.csv" se encuentran detalles concretos de la ejecución capa por capa como por ejemplo el ciclo en el que empiezan y acaban las lecturas y escrituras de DRAM para IFMAP FILTER y OFMAP o el volumen de tamaño en bytes de las tres matrices IFMAP FILTER y OFMAP.

"_cycles.csv" contiene información sobre los ciclos que ha tardado la ejecución de cada capa que forma la red ejecutada y el porcentaje de utilización medio a lo largo de todos los ciclos que ha tenido el array de PE durante la ejecución de dicha capa. A continuación, se presenta un ejemplo visual:

	A	B	C	D	E
1	Layer		Cycles		% Utilization
2	Conv		181160		97.630638
3	Res_conv1		2727944		97.63483649
4	Res_conv2		2727944		97.63483649
5	ValueHead_conv		11777		12.26222826
6	ValueHead_FC1		11560		12.48864521
7	ValueHead_FC2		257		1.5625
8	PolicyHead_Conv		11778		24.52224145
9	PolidyHead_FC		33214		12.29577838
10					

Figura 15: Ejemplo real sobre un archivo "_cycles.csv"

Por último, en el directorio “layer_wise” se encuentran los ficheros correspondientes a las trazas de lectura y escritura DRAM y lectura y escritura SRAM resultantes de la ejecución de cada capa tal y como se muestra a continuación.

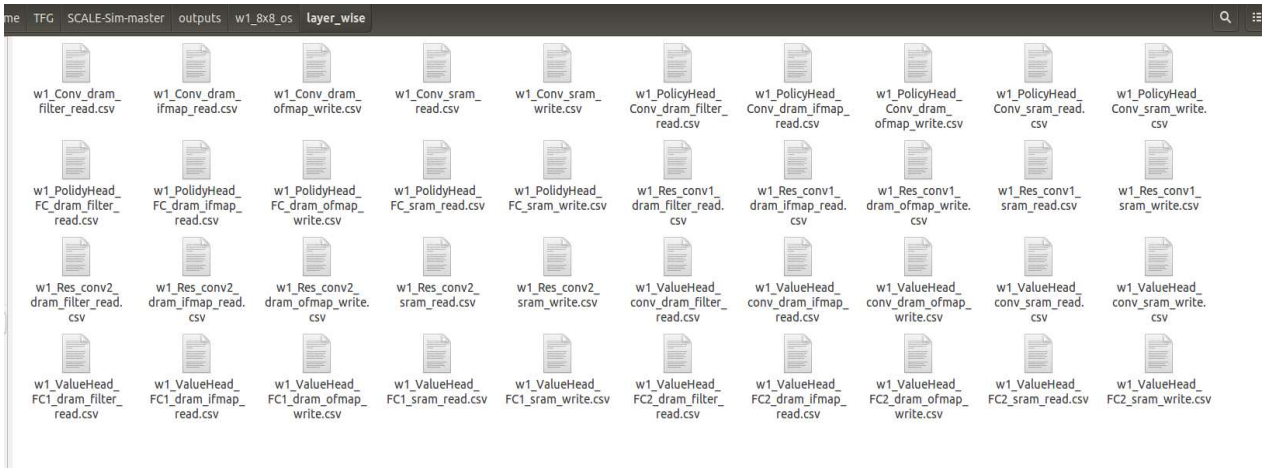

Figura 16: Ejemplo real sobre el directorio "layer_wise"

Los nombres de los ficheros son bastante representativos todos empezando por el nombre la carga y seguidos de la capa a la que corresponden los datos que contiene y finalmente el tipo de memoria y si es lecturas o escrituras a esa memoria.

En este trabajo no se van a mostrar el contenido de todas y cada una de ellas, pero sí que se va a mostrar el contenido de los archivos de lectura y escritura del triple buffer ya que son muy representativas y didácticas para explicar y demostrar el funcionamiento del simulador y también son las trazas que han sido empleadas en el proyecto para el cálculo de la energía que consume el simulador en las ejecuciones realizadas para el *CAPÍTULO 6*.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	0	0								10000000									
2	1	1	17							10000001	10000153								
3	2	2	18	34						10000002	10000154	10000306							
4	3	3	19	35	51					10000003	10000155	10000307	10000459						
5	4	4	20	36	52	68				10000004	10000156	10000308	10000460	10000612					
6	5	5	21	37	53	69	85			10000005	10000157	10000309	10000461	10000613	10000765				
7	6	6	22	38	54	70	86	102		10000006	10000158	10000310	10000462	10000614	10000766	10000918			
8	7	7	23	39	55	71	87	103	119	10000007	10000159	10000311	10000463	10000615	10000767	10000919	10001071		
9	8	8	24	40	56	72	88	104	120	10000008	10000160	10000312	10000464	10000616	10000768	10000920	10001072		
10	9	9	25	41	57	73	89	105	121	10000009	10000161	10000313	10000465	10000617	10000769	10000921	10001073		
11	10	10	26	42	58	74	90	106	122	10000010	10000162	10000314	10000466	10000618	10000770	10000922	10001074		
12	11	11	27	43	59	75	91	107	123	10000011	10000163	10000315	10000467	10000619	10000771	10000923	10001075		
13	12	12	28	44	60	76	92	108	124	10000012	10000164	10000316	10000468	10000620	10000772	10000924	10001076		
14	13	13	29	45	61	77	93	109	125	10000013	10000165	10000317	10000469	10000621	10000773	10000925	10001077		
15	14	14	30	46	62	78	94	110	126	10000014	10000166	10000318	10000470	10000622	10000774	10000926	10001078		
16	15	15	31	47	63	79	95	111	127	10000015	10000167	10000319	10000471	10000623	10000775	10000927	10001079		
17	16	16	32	48	64	80	96	112	128	10000016	10000168	10000320	10000472	10000624	10000776	10000928	10001080		
18	17	17	33	49	65	81	97	113	129	10000017	10000169	10000321	10000473	10000625	10000777	10000929	10001081		
19	18	18	34	50	66	82	98	114	130	10000018	10000170	10000322	10000474	10000626	10000778	10000930	10001082		
20	19	19	35	51	67	83	99	115	131	10000019	10000171	10000323	10000475	10000627	10000779	10000931	10001083		
21	20	20	36	52	68	84	100	116	132	10000020	10000172	10000324	10000476	10000628	10000780	10000932	10001084		
22	21	21	37	53	69	85	101	117	133	10000021	10000173	10000325	10000477	10000629	10000781	10000933	10001085		
23	22	22	38	54	70	86	102	118	134	10000022	10000174	10000326	10000478	10000630	10000782	10000934	10001086		
24	23	23	39	55	71	87	103	119	135	10000023	10000175	10000327	10000479	10000631	10000783	10000935	10001087		
25	24	24	40	56	72	88	104	120	136	10000024	10000176	10000328	10000480	10000632	10000784	10000936	10001088		
26	25	25	41	57	73	89	105	121	137	10000025	10000177	10000329	10000481	10000633	10000785	10000937	10001089		
27	26	26	42	58	74	90	106	122	138	10000026	10000178	10000330	10000482	10000634	10000786	10000938	10001090		
28	27	27	43	59	75	91	107	123	139	10000027	10000179	10000331	10000483	10000635	10000787	10000939	10001091		
29	28	28	44	60	76	92	108	124	140	10000028	10000180	10000332	10000484	10000636	10000788	10000940	10001092		
30	29	29	45	61	77	93	109	125	141	10000029	10000181	10000333	10000485	10000637	10000789	10000941	10001093		
31	30	30	46	62	78	94	110	126	142	10000030	10000182	10000334	10000486	10000638	10000790	10000942	10001094		

Figura 17: Ejemplo real sobre un archivo "_sram_read.csv"

Todos los ficheros de trazas siempre tienen en la primera columna el ciclo de la ejecución simulada en el que tuvo lugar el acceso y las siguientes columnas representan las direcciones accedidas (cada celda representa un pixel leído o escrito). Las direcciones de memoria se calculan a partir de parámetros definidos por el usuario que ya han sido comentados en la sección 5.1 de este capítulo.

La simulación realizada tenía configurados los parámetros de direcciones base igual que en la *Figura 10.* Por tanto, se puede identificar fácilmente que las ocho primeras columnas desde la izquierda sin contar la columna de los ciclos representan lecturas de IFMAP y las ocho siguientes lecturas de FILTER. También se puede ver cómo las lecturas y escrituras realizadas coinciden con el funcionamiento del flujo de datos OS explicado a partir de la Figura 8 al inicio de este capítulo. En el primer ciclo entra 1 píxel de IFMAP y uno de FILTER al array, en el segundo ciclo dos, en el tercer ciclo 3 etc. En el ciclo 8 las 8 filas y columnas del array ya reciben píxeles y por tanto ya no se pasará de esa cifra de lecturas de cada tipo de dato en cada ciclo, pues hay que recordar que el ejemplo expuesto se basa en una matriz 8x8 como ya se ha mencionado.

	A	B	C	D	E	F	G	H	I	J
1	160	20000000	20000001	20000002	20000003	20000004	20000005	20000006	20000007	
2	161	20000256	20000257	20000258	20000259	20000260	20000261	20000262	20000263	
3	162	20000512	20000513	20000514	20000515	20000516	20000517	20000518	20000519	
4	163	20000768	20000769	20000770	20000771	20000772	20000773	20000774	20000775	
5	164	20001024	20001025	20001026	20001027	20001028	20001029	20001030	20001031	
6	165	20001280	20001281	20001282	20001283	20001284	20001285	20001286	20001287	
7	166	20001536	20001537	20001538	20001539	20001540	20001541	20001542	20001543	
8	167	20001792	20001793	20001794	20001795	20001796	20001797	20001798	20001799	
9	313	20002048	20002049	20002050	20002051	20002052	20002053	20002054	20002055	
10	314	20002304	20002305	20002306	20002307	20002308	20002309	20002310	20002311	
11	315	20002560	20002561	20002562	20002563	20002564	20002565	20002566	20002567	
12	316	20002816	20002817	20002818	20002819	20002820	20002821	20002822	20002823	
13	317	20003072	20003073	20003074	20003075	20003076	20003077	20003078	20003079	
14	318	20003328	20003329	20003330	20003331	20003332	20003333	20003334	20003335	
15	319	20003584	20003585	20003586	20003587	20003588	20003589	20003590	20003591	
16	320	20003840	20003841	20003842	20003843	20003844	20003845	20003846	20003847	
17	466	20004096	20004097	20004098	20004099	20004100	20004101	20004102	20004103	
18	467	20004352	20004353	20004354	20004355	20004356	20004357	20004358	20004359	
19	468	20004608	20004609	20004610	20004611	20004612	20004613	20004614	20004615	
20	469	20004864	20004865	20004866	20004867	20004868	20004869	20004870	20004871	
21	470	20005120	20005121	20005122	20005123	20005124	20005125	20005126	20005127	
22	471	20005376	20005377	20005378	20005379	20005380	20005381	20005382	20005383	
23	472	20005632	20005633	20005634	20005635	20005636	20005637	20005638	20005639	
24	473	20005888	20005889	20005890	20005891	20005892	20005893	20005894	20005895	
25	619	20006144	20006145	20006146	20006147	20006148	20006149	20006150	20006151	
26	620	20006400	20006401	20006402	20006403	20006404	20006405	20006406	20006407	
27	621	20006656	20006657	20006658	20006659	20006660	20006661	20006662	20006663	
28	622	20006912	20006913	20006914	20006915	20006916	20006917	20006918	20006919	

Figura 18: Ejemplo real sobre un archivo "_sram_write.csv"

De igual modo que en el ejemplo anterior se identificaban los tipos de datos leídos en base al archivo de configuración, en este ejemplo vemos como sigue las mismas reglas, mostrando únicamente las escrituras de OFMAP tal y como anunciaba el nombre del fichero.

El simulador calcula el ciclo en el cual se realizará la primera escritura en el buffer de OFMAP cómo el número de columnas + ventana convolucional – 1.

Al valor resultante se le resta uno debido a que el simulador empieza a contar ciclos desde el 0.

La ventana convolucional es el número de píxeles de IFMAP que se necesitan para computar un único pixel de OFMAP.

La facilidad para identificar a qué tipo de datos (entrada pesos o salida) y a qué partición de la memoria pertenece cada dato leído o escrito en las trazas ha sido aprovechada en el diseño de los scripts que se exponen en la siguiente sección.

5.4 Desarrollo de scripts

Para llevar a llevar a cabo este proyecto ha sido necesario la realización de una gran cantidad de simulaciones en SCALE-Sim, simulador que realiza las ejecuciones de una en una individualmente sin poder hacer una lista o encolar ejecuciones. Para optimizar el proceso de lanzamiento de ejecuciones se ha creado un script en Bash que permite el lanzamiento de un conjunto de ejecuciones definidas por el usuario. Este script requiere de una estructura fija de directorios para su correcto funcionamiento. De este modo el script da la opción de realizar ejecuciones rectangulares (ej. 32x64) o cuadradas (ej. 64 x 64) respecto al tamaño del array que se desea simular. El script también recibe del usuario desde qué tamaño empezar y de cuánto se desea que sea el incremento para la siguiente ejecución. En otras palabras, si queremos ejecutar una carga variando los tamaños del array de PE desde 8 hasta 32 le indicaremos al simulador que queremos empezar desde 8 con incrementos de 2 (x2) hasta llegar a 32.

El script está diseñado para no tener que indicar las cargas como parámetro de entrada cada vez que se desea lanzar una ejecución, en vez de eso el script lee las cargas desde la w1 hasta la w7 (vistas en la *sección 5.2)* desde un directorio que debemos crear en el directorio ya existente "topologies" del simulador con el nombre "fig6ws" y poner las distintas cargas con el nombre apropiado del archivo desde w1.csv hasta w7.csv y ejecutando el simulador con ellas.

Como ya vimos las entradas al simulador no sólo eran las cargas, también un archivo de configuración con extensión ".cfg" que configuraba la arquitectura del acelerador. El script genera un nuevo archivo de configuración para cada ejecución con los parámetros que el usuario establece de forma automática y sin tener que manipular directamente los ficheros antes de cada ejecución. Para ello se han de crear dos directorios dentro del ya existente llamado "configs", la primera carpeta se llamará "fig6cfgsR" albergará los archivos de configuración de matrices rectangulares si es el caso y la segunda se llamará "fig6cfgs" y albergará los archivos de configuración que cree el script en caso de matrices cuadradas.

Además, de esto el script debe estar ubicado en la carpeta del simulador.

El script cuenta con un comando de ayuda que nos muestra sus opciones y los parámetros que se pueden configurar en orden. Para ello se ha de ejecutar ./script.sh --help y se mostrará en pantalla la siguiente información:

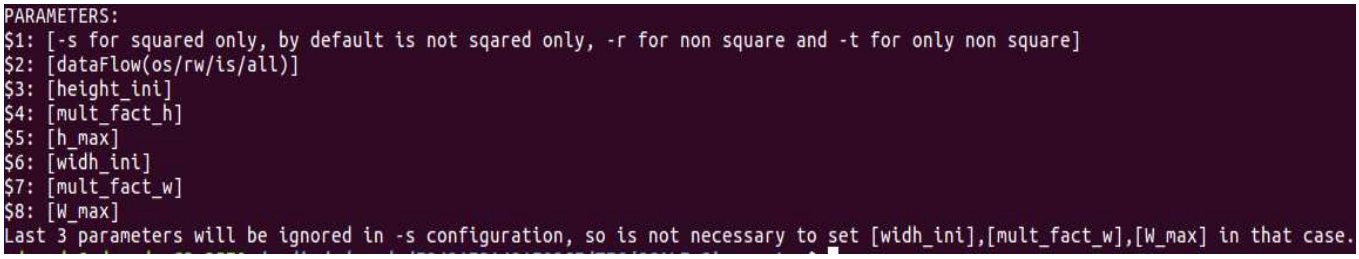

Figura 19: Texto de ayuda de "script.sh"

Si se desea ampliar información acerca de este script, el código se encuentra disponible en el *ANEXO 1*.

Una vez obtenidos los resultados de las ejecuciones, fue necesaria la elaboración de otro script para automatizar el proceso de contar el número de accesos al buffer de IFMAP, al de FILTER y al de OFMAP. Para obtener estos valores se debía entrar en el directorio “layer_wise” del que ya se ha hablado en la sección anterior y recorrer todos los archivos “_sram_read.csv” y “sram_write.csv”. Este proceso se debe realizar para los resultados de cada simulación.

Debido a esta necesidad se elaboró el script “count.sh” cuyo código está disponible en el *ANEXO 2*.

El script recorre cada directorio de resultados de cada simulación entrando en “layer_wise” y después en los archivos de trazas de lectura y escritura de cada capa, sumando los accesos a cada uno de los buffers y acumulando el valor en 3 contadores individuales. El script muestra por pantalla el contador para cada capa y cuando termina con el directorio de resultados de una simulación escribe la suma total de accesos a cada uno de los buffers durante toda la ejecución que se simuló en un fichero llamado “output_access.out”. A continuación, un ejemplo:

```
Output: w1_32x64_os
Total SRAM_IFMAP reads during execution: 5694580
Total SRAM_FILTER reads during execution: 12551412
Total SRAM_OFMAP write during execution: 223654

Output: w1_64x32_os
Total SRAM_IFMAP reads during execution: 11204072
Total SRAM_FILTER reads during execution: 6452724
Total SRAM_OFMAP write during execution: 223654

Output: w2_32x64_os
Total SRAM_IFMAP reads during execution: 67795080
Total SRAM_FILTER reads during execution: 55125088
Total SRAM_OFMAP write during execution: 1006077

Output: w2_64x32_os
Total SRAM_IFMAP reads during execution: 67795080
Total SRAM_FILTER reads during execution: 27730272
Total SRAM_OFMAP write during execution: 1006077
```

Figura 20: Ejemplo de "output_access.out "

Cómo se puede ver el archivo contiene el nombre de la simulación y los accesos totales a cada uno de los tres buffers.

Para que el script pueda funcionar correctamente debe estar ubicado en la carpeta del simulador.

6 CAPÍTULO: Resultados experimentales

En este capítulo se exponen los resultados obtenidos de las ejecuciones que se han realizado a lo largo del tiempo del proyecto configurando los ficheros explicados en las *secciones 5.1* y *5.2* y trabajando con los datos resultantes de éstas, contenidos en los ficheros explicados también en la *sección 5.3*, todas ellas en el capítulo anterior. En este capítulo se analizan los resultados y comparan entre ellos para tratar de extraer conocimiento. Todos los experimentos se realizaron con los tamaños de buffers: IFMAP 512KB, FILTER 512KB y OFMAP 256KB además del flujo de datos OS.

6.1 Estudio de las cargas utilizadas

Antes de comenzar el análisis y a fin de ayudar al lector a una mejor interpretación del lector de las gráficas, se proporciona una breve descripción de las cargas empleadas en este proyecto.

Las distintas cargas, desde la W1 hasta la W7 son un conjunto de cargas bastante diferenciadas entre sí y variadas, y cubren un espectro muy amplio, lo que hace que los resultados y las conclusiones de una experimentación con todas ellas puedan considerarse para la generalización tentativa de las conclusiones y extensión a un conjunto de redes neuronales más amplio que las siete cargas, siempre recordando el dominio de las cargas, que deben ser CNN o fully connected como ya se ha hablado.

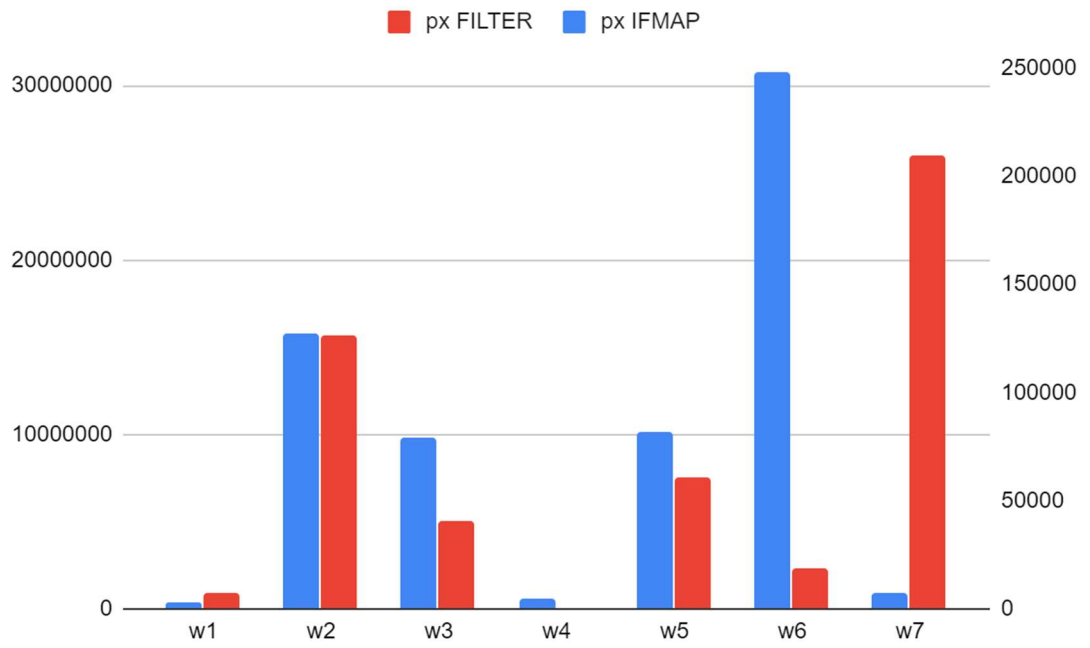

Figura 21: Número de píxeles de las matrices IFMAP y FILTER de cada carga.

Esta gráfica que muestra información sobre las matrices de cada carga mediante el número de píxeles totales de FILTER y IFMAP de éstas, lo que permite visualizar de forma muy aproximada la diversidad en las cargas empleadas en el proyecto, aunque no hay que olvidar que también existen otros parámetros respecto a la distribución de esos píxeles (número de matrices y dimensiones de éstas) que no se ven representadas en el gráfico, pero que no son necesarios ilustrar pues el gráfico sólo ya demuestra lo mencionado.

De este modo tenemos desde cargas muy ligeras como las W1 y W4 a cargas muy pesadas como las W2, W6 y W7.

Otro parámetro no directamente relacionado con las matrices que forman la red es el número de canales, parámetro muy importante ya que cada capa supone una ejecución distinta dentro del acelerador.

Carga	Número de capas
w1	10
w2	7
w3	47
w4	11
w5	56
w6	6
w7	894

Tabla 2:Número de capas de cada carga

Esta tabla le aporta una segunda dimensión a la demostración de la diversidad en las cargas mostrada en la figura anterior. Ambas se pueden observar para obtener una visión bastante amplia sobre cómo es cada carga y entender posteriormente su comportamiento en las simulaciones y los resultados. Por ejemplo, se puede observar que w6, a pesar de ser una de las cargas con más volumen de cálculo, es la que menos capas tiene lo que significa que estamos ante una carga con pocas capas pero muy pesadas. Otro ejemplo puede ser w7, esta carga está formada por muchas capas poco pesadas pero que sumadas hacen que la carga sea una de las que mayor volumen de cálculo tenga.

6.2 Tiempos de ejecución

Uno de los parámetros más importantes en cuanto a prestaciones que resulta de interés medir es el tiempo de ejecución y cómo varía frente a variaciones en los parámetros de la arquitectura del acelerador. A continuación, se exponen los resultados de variaciones en el tiempo de ejecución para las cargas del proyecto ya mencionadas frente a variaciones en las dimensiones de la matriz de PE para el flujo de datos OS.

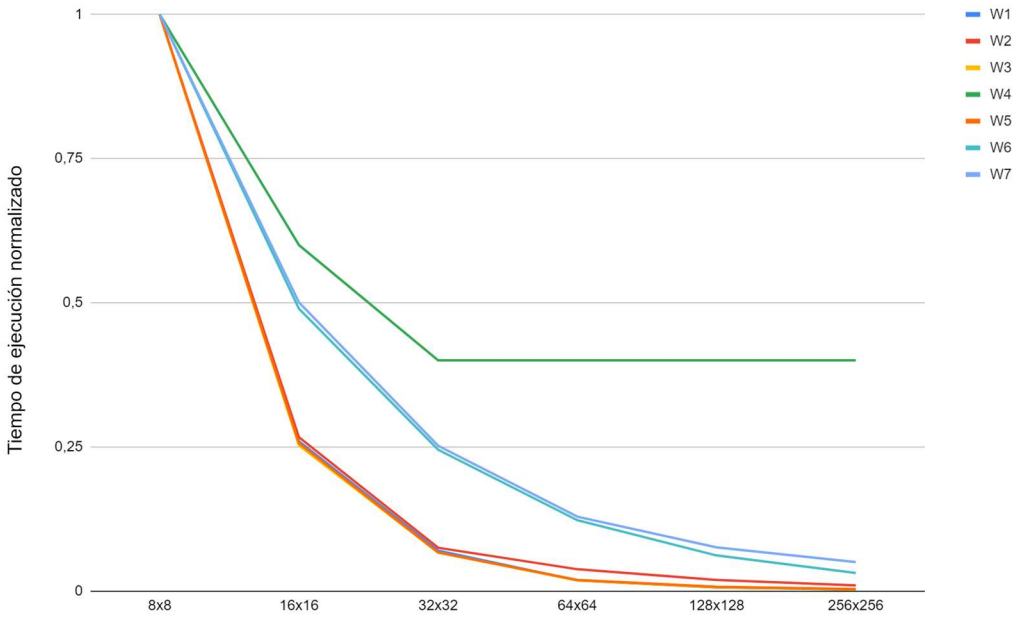

Figura 22: Tiempo de ejecución frente a variaciones uniformes del tamaño del array

En esta gráfica se puede observar la sensibilidad del tiempo de ejecución de las distintas cargas frente a variaciones del tamaño del array con geometría cuadrada. En general, se presentan relaciones logarítmicas en las cuales a partir del tamaño 64x64 la mejora que produce en el tiempo de ejecución el aumento de silicio dedicado al procesamiento deja de ser interesante respecto a la progresión de mejoras de los tamaños anteriores.

Para afinar aún más en esta afirmación se utiliza un enfoque distinto a través de una gráfica de EDP (Energy Delay Product).

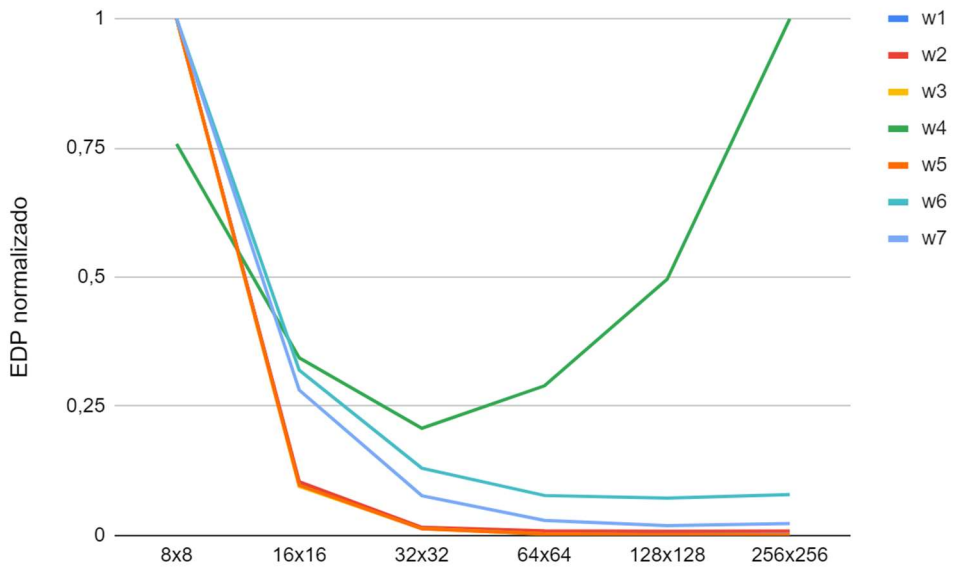

Figura 23: EDP frente a variaciones uniformes del tamaño del array

Esta gráfica nos permite medir conjuntamente la mejora del tiempo de ejecución y el consumo energético que introducen las variaciones en el dimensionamiento anteriormente expuestas. Gracias a ella podemos afirmar, esta vez sí, que para seis de las siete cargas la configuración 64x64 es la que mejor rendimiento relativo al coste ofrece para la ejecución en OS.

En los experimentos anteriores los incrementos en el número total de PE con cada nuevo tamaño eran de x4 debido a la geometría cuadrada, a continuación, se exponen unas gráficas con geometría rectangular para valorar las variaciones en el tiempo de ejecución de las cargas con incrementos de x2 en altura o anchura.

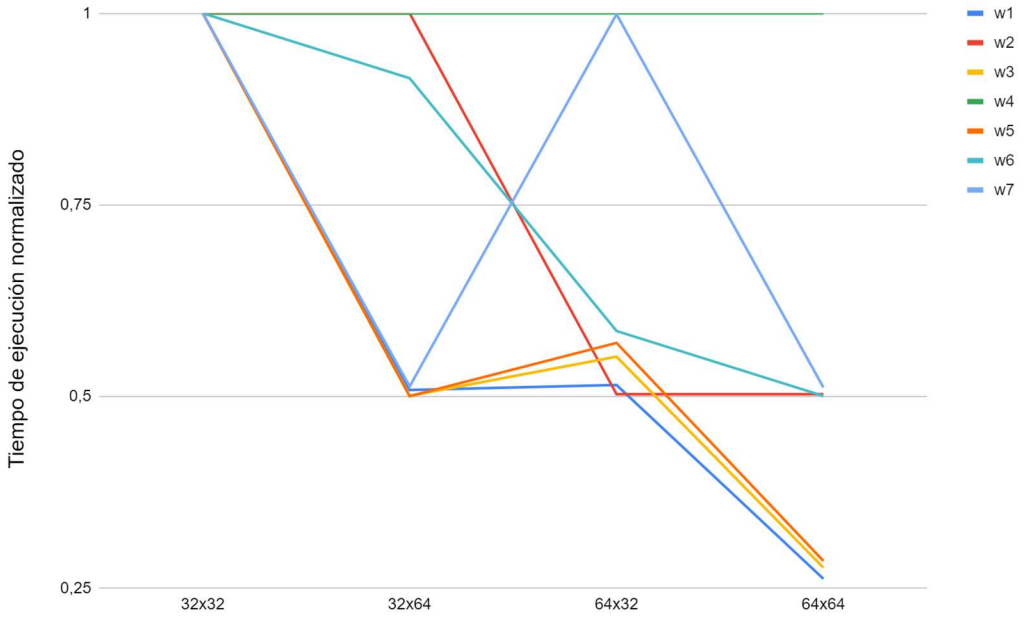

Figura 24: Tiempo de ejecución frente a variaciones irregulares del tamaño del array

Para estos experimentos se han elegido tamaños rectangulares del array entre las dos configuraciones que ofrecían una relación rendimiento/tamaño más interesante, estos son 32x32 y 64x64 en busca un incremento cercano lo más cercano posible al de 64x64, pero con la mitad de PE.

Los resultados que encontramos en este caso son distintos al del primer conjunto de experimentos con distribuciones uniformes. El rendimiento que se obtiene en distribuciones irregulares varía en función de las características de la carga ejecutada, por lo tanto, en este caso no es posible encontrar de manera clara la mejor configuración. La mejor aproximación que se puede realizar es que para la configuración 32x64, cuatro de las siete cargas mejoran bajando su tiempo de ejecución un 50% y en la configuración 64x32 cinco de 7 cargas ofrecen una mejora en el tiempo de ejecución cercanas al 50%.

Al igual que en el primer experimento, para validar las conclusiones sobre las variaciones en geometría rectangular se emplea de nuevo la gráfica EDP.

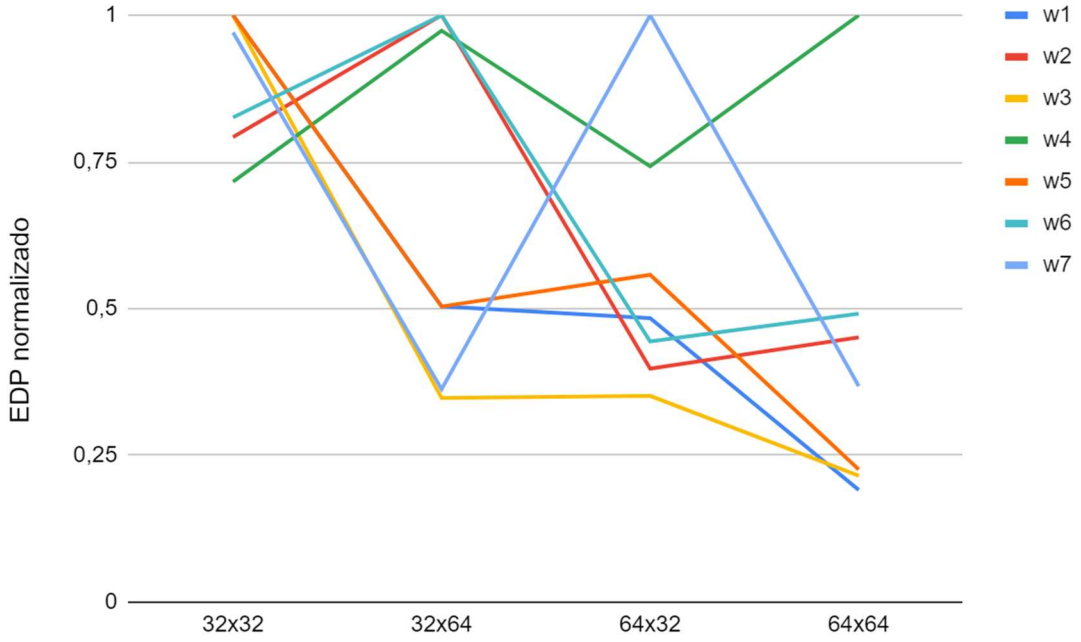

Figura 25: EDP frente a variaciones irregulares del tamaño del array

En este caso la gráfica EDP nos muestra cómo existe una mayor concentración de puntos cercanos al 0,5 (este valor significa que el tiempo de ejecución ha bajado a la mitad) en la configuración 64x32 que en la configuración 32x64 ofreciendo por tanto esta primera una relación entre número de PE, mejora del tiempo de ejecución y consumo energético mejor que la segunda para un mayor espectro de cargas y con valores competitivos frente a la configuración 64x64 con la mitad de PE que ésta.

La causa del hecho de que el tiempo de ejecución para distribuciones irregulares varía en función de las características de la carga ejecutada y no sea una progresión lineal de mejora como en configuraciones uniformes se expone en la sección siguiente de la mano de otro parámetro muy importante cómo es el porcentaje de utilización de los elementos de procesamiento del array durante la ejecución.

6.3 Porcentajes de utilización

El porcentaje medio de utilización se calcula como la suma del porcentaje del array que está siendo utilizada en cada ciclo y después esta suma es dividida entre el número total de ciclos. Esta métrica nos permite saber si una configuración está sobredimensionada para la ejecución de una carga o si por el contrario con un mayor número de PE se lograrían mejores prestaciones. Tiempo de ejecución y porcentaje de utilización están directamente relacionados.

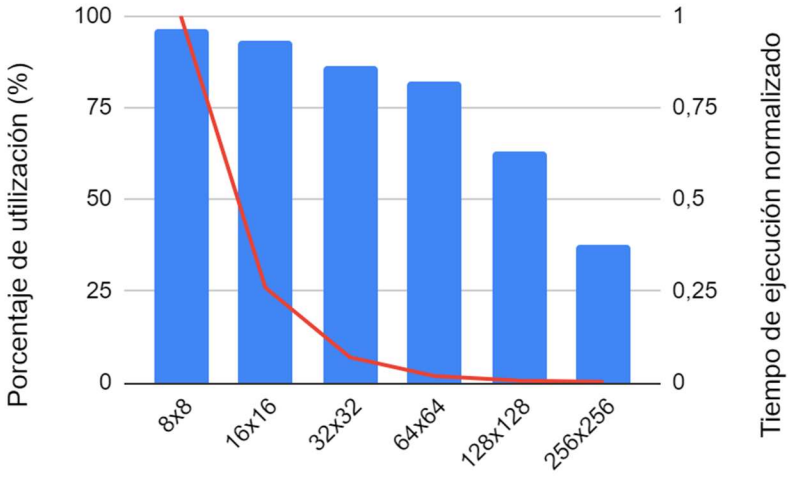

Figura 26: Porcentaje de utilización y tiempo de ejecución para W1(uniformes)

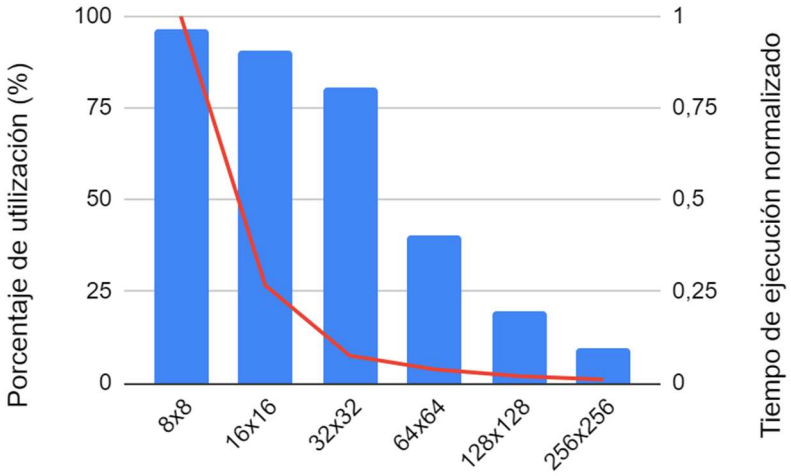

Figura 27: Porcentaje de utilización y tiempo de ejecución para W2(uniformes)

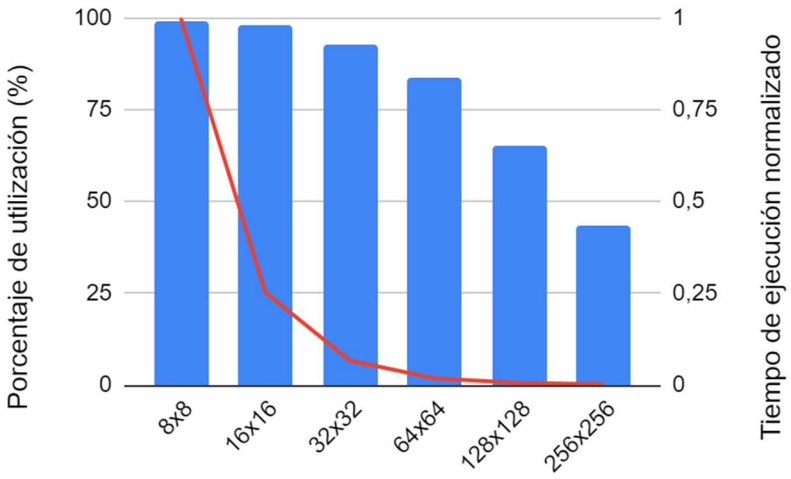

Figura 28: Porcentaje de utilización y tiempo de ejecución para W3(uniformes)

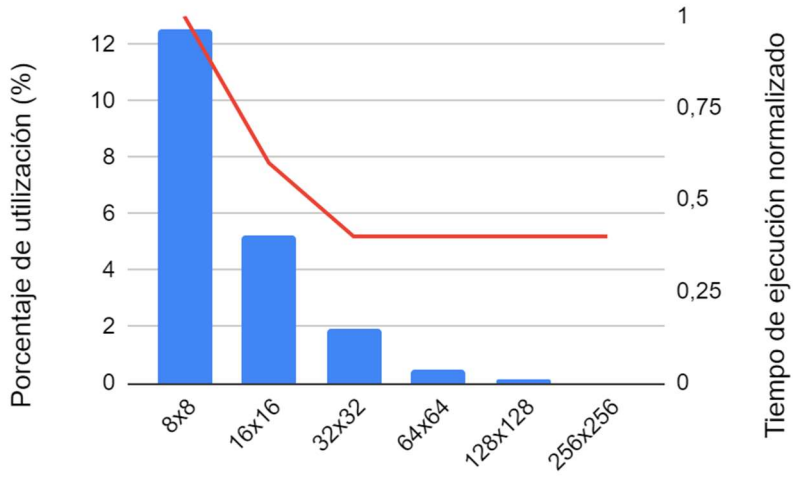

Figura 29: Porcentaje de utilización y tiempo de ejecución para W4(uniformes)

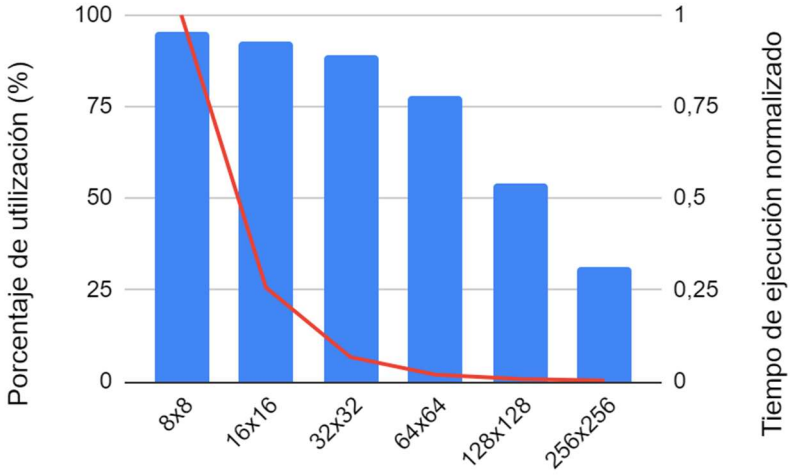

Figura 30: Porcentaje de utilización y tiempo de ejecución para W5(uniformes)

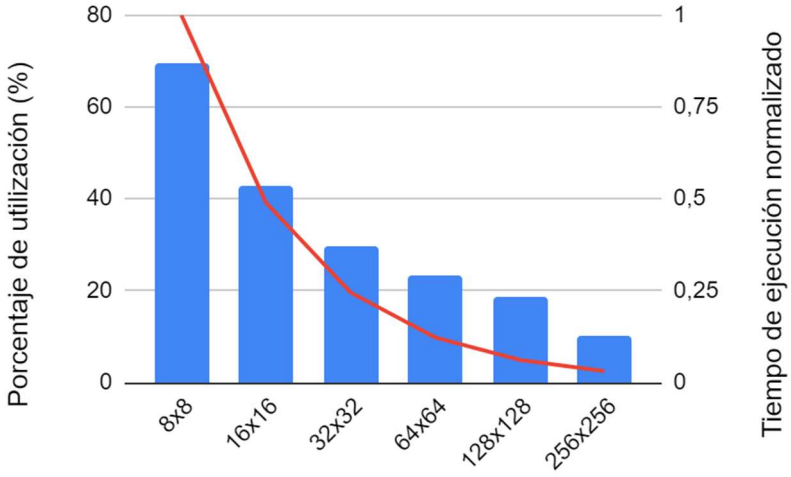

Figura 31: Porcentaje de utilización y tiempo de ejecución para W6(uniformes)

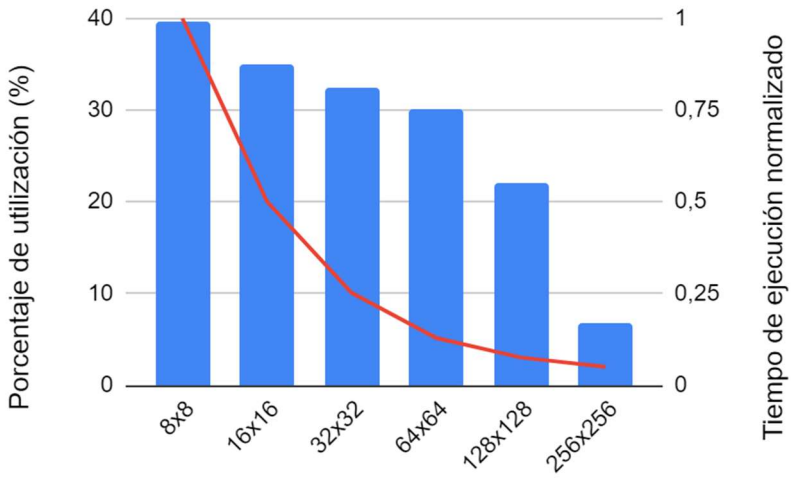

Figura 32: Porcentaje de utilización y tiempo de ejecución para W7 (uniformes)

Mientras exista un margen de mejora, el tiempo de ejecución es directamente proporcional al porcentaje de utilización en condiciones ideales como las que siempre contempla el simulador. Esto se comprueba en las W1, W2, W3, W5, W6 y W7, pero no en la W4. Esto se puede explicar mediante la Ley de Amdahl. En base a esta ley se entiende que la parte del tiempo de ejecución que dependía del dimensionamiento del array de PE alcanza su valor más bajo a partir de 32x32 y aumentos en el tamaño del array no implicarán mejoras en el tiempo de ejecución a partir de ese tamaño.

A partir de ahora y hasta el final de esta sección cuando se haga referencia a un mínimo en tiempo de ejecución se estará haciendo referencia al mínimo tiempo de ejecución alcanzable si la mejora en el porcentaje del tiempo de ejecución que depende del dimensionamiento del array es máxima, de acuerdo con la Ley de Amdahl cómo se ha mencionado anteriormente.

Aunque en un primer momento ver en las gráficas cómo con disminuciones del porcentaje de utilización mejora el tiempo de ejecución pueda parecer extraño, hay que tener dos cosas en cuenta, la primera es que cada carga tiene unos requerimientos distintos de hardware para optimizar sus ejecuciones, algunas de ellas requerirán un mayor número de columnas y otras de ellas por el contrario un mayor número de filas de array (hay que recordar que se inyecta a las filas los IFMAP y a las columnas los FILTER) y como se ha visto en la sección 6.1 este proyecto emplea cargas muy variadas para poder aplicar las conclusiones a un espectro amplio de redes CNN como ya se ha dicho.

La segunda cosa a tener en cuenta es que las siete gráficas anteriores son referentes al experimento en arrays de geometría cuadrada o de distribución uniforme y por lo tanto se incrementan columnas y filas a la vez, por lo tanto, estamos hablando de configuraciones bastante

generales y algunas cargas cuyo tiempo de ejecución óptimo se alcance con una configuración con muchas filas y pocas columnas el porcentaje de utilización será bajo.

Una red que alcance la mejora máxima del tiempo de ejecución que depende del dimensionamiento del array con 256 filas y 16 columnas, al aumentar uniformemente las dimensiones del array estamos aumentando un número de columnas que no van a ser empleadas en la ejecución, por tanto, el porcentaje de utilización disminuirá con cada incremento del tamaño del array hasta llegar a 256x256 donde el porcentaje de utilización será bajo. A pesar de esto en 256x256 el tiempo de ejecución se habrá reducido hasta su mínimo, aunque con un gran desaprovechamiento de recursos.

En las gráficas se puede observar todo eso. Las cargas que presentan una mayor sensibilidad en el porcentaje de utilización del array conforme aumenta su tamaño uniformemente son la W2, W6 y W7 sin contar la W4 cuyo caso particular ya ha sido discutido.

En W2 es especialmente visible el brusco decremento en el porcentaje de utilización que sufre al pasar de 32x32 a 64x64, por lo tanto, o bien un valor cercano a 32 es el tamaño óptimo de altura o bien de anchura.

Estas cargas por tanto barruntan pertenecer al caso explicado en el ejemplo anterior, y se puede pensar en un primer momento que requieren una altura notablemente mayor que la anchura del array o viceversa para alcanzar el mínimo tiempo de ejecución posible.

Por el contrario, las cargas W1, W3 y W5 presentan una menor sensibilidad en el porcentaje de utilización con los aumentos en el tamaño del array, por tanto, se puede establecer la hipótesis de que la configuración óptima de estas cargas no tiene una gran diferencia entre la anchura y la altura del array.

Para comprobar las dos hipótesis planteadas es necesaria la experimentación con distribuciones irregulares de PE en el array.

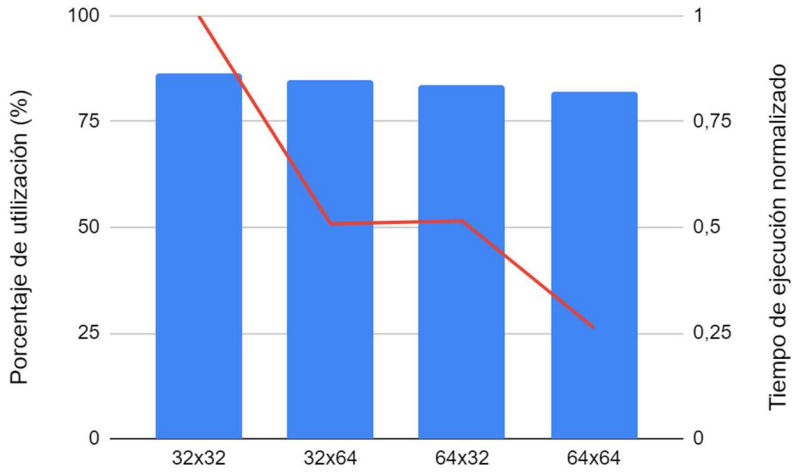

Figura 33: Porcentaje de utilización y tiempo de ejecución para W1 (irregulares)

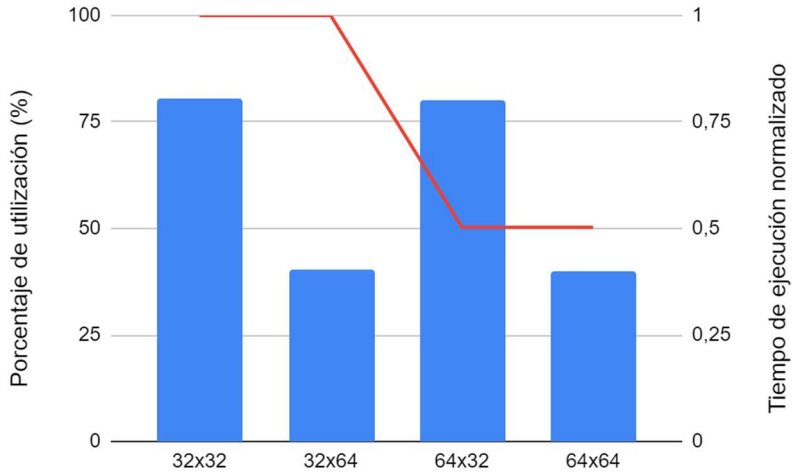

Figura 34: Porcentaje de utilización y tiempo de ejecución para W2 (irregulares)

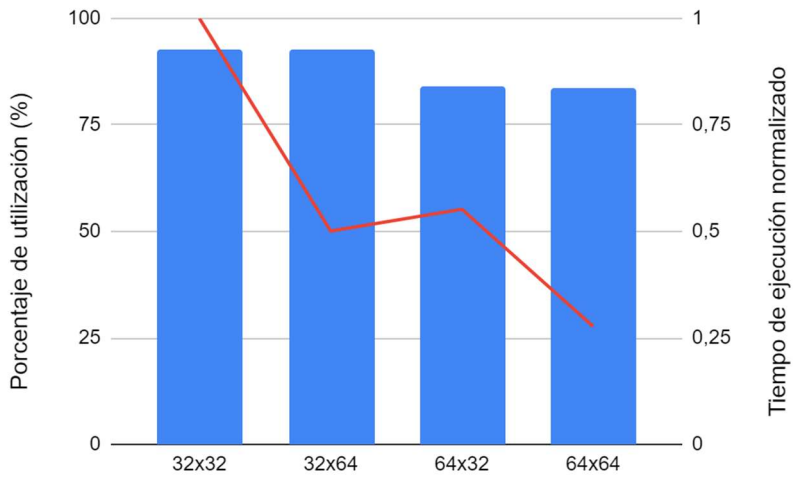

Figura 35: Porcentaje de utilización y tiempo de ejecución para W3 (irregulares)

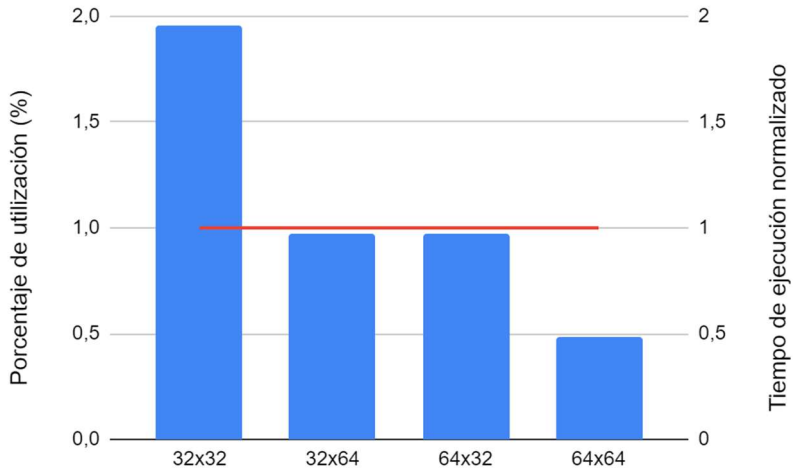

Figura 36: Porcentaje de utilización y tiempo de ejecución para W4 (irregulares)

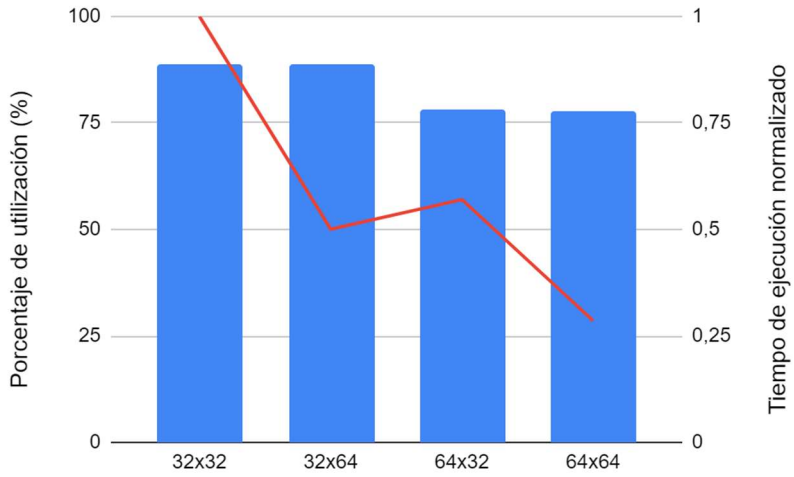

Figura 37: Porcentaje de utilización y tiempo de ejecución para W5 (irregulares)

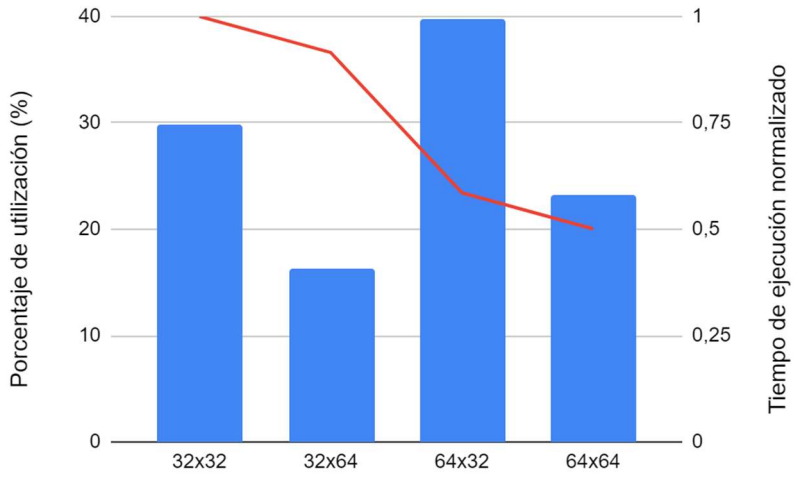

Figura 38: Porcentaje de utilización y tiempo de ejecución para W6 (irregulares)

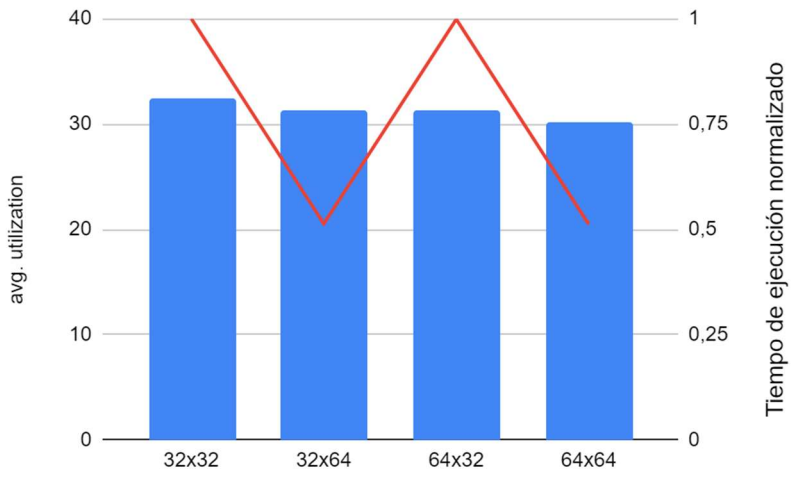

Figura 39: Porcentaje de utilización y tiempo de ejecución para W7 (irregulares)

Las gráficas sobre los experimentos de configuraciones rectangulares apoyan la hipótesis planteada anteriormente. Se puede observar cómo las cargas W1, W3 y W5 no presentan mejoras significativas al duplicar únicamente la altura o anchura del array. La carga W7 por el contrario, aunque aparenta no presentar el comportamiento esperado, en 32x64 con un aumento de utilización respecto a 64x32 del 0,1% apenas imperceptible en la gráfica supone una mejora del tiempo de ejecución considerable, bajando éste hasta la mitad respecto a las configuraciones 32x32 y 64x32. Por tanto, estamos ante una carga que con un incremento de la anchura x2 presenta un incremento en el porcentaje de utilización, pero con un gran impacto positivo en el tiempo de ejecución.

La carga W2 cumple exactamente el supuesto comentado a partir de las gráficas de ejecuciones uniformes, recordando lo comentado se sospechaba que entre 32x32 y 64x64 se encontraba la clave para averiguar si las filas o las columnas del array en esta carga alcanzaban el máximo de utilización posible.

La *figura 34* muestra claramente cómo el aumento del número de columnas de 32 a 64 produce un descenso del porcentaje de utilización a la mitad, esto quiere decir que las nuevas 32 columnas introducidas no fueron aprovechadas en la ejecución, en cambio al aumentar el número de filas de 32 a 64 el porcentaje de utilización aumenta el doble respecto la configuración anterior. Esto indica que sucede todo lo contrario, en este caso todas las nuevas filas introducidas son aprovechadas en la ejecución. Un comportamiento similar se observa en la carga W6 a diferencia de que un porcentaje modesto de las columnas introducidas sí son aprovechadas. En la carga W4 se confirma que en la configuración 32 x 32 encuentra su configuración más óptima y que

aumentos tanto en el número de columnas y/o en el número de filas producen una caída en el porcentaje de utilización proporcional, sin ningún efecto en el tiempo de ejecución.

Estas explicaciones dan sentido a lo observado en la sección 6.2 donde la gráfica sobre tiempos de ejecución en los experimentos relativos a configuraciones irregulares (*figura 24*) parecían no tener un patrón aparente y que dependía dependiendo de la carga unas configuraciones eran mejor que otras, pero no se apreciaba el por qué. Ahora tras poner el porcentaje de utilización sobre la mesa junto al tiempo de ejecución de cada carga se ha explicado de qué depende esta arbitrariedad.

6.4 Consumos energéticos

En la sección 6.1 se hablaba ya de que relacionaba los datos obtenidos sobre el tiempo de ejecución y sobre el consumo. En esta sección vamos a analizar de forma aislada las variaciones en el consume energético ante variaciones en el dimensionamiento del array con datos de energía obtenidos a partir del modelado de los buffers en el simulador CACTI 6.0 en su configuración por defecto, variando los parámetros que se muestran en la siguiente tabla:

Parámetro	**Valor**
size (bytes)	Tamaño deseado
block size (bytes)	*
associativity	1
output/input bus width	block size x 8
exclusive read port	1
exclusive write port	1
access mode	normal
cache type	SRAM
Optimize ED or ED^2	ED
Cache model	UCA

Tabla 3: Parámetros modificados en CACTI6.0 para modelar los buffers

* (ancho del array para FILTER y OFMAP o alto para IFMAP) x tamaño de datos en bytes. Los datos empleados son de 1byte (tamaño de pixel).

Esta configuración modela un buffer FIFO del tamaño indicado, con. El fichero de configuración usado para CACTI. En el *ANEXO 3* se dispone como ejemplo de las configuraciones realizadas con un archivo de configuración del simulador referente al buffer SRAM de IFMAP para una arquitectura de acelerador con array 8x8.

Estas gráficas sobre el consumo nos ayudarán a entender mejor las gráficas de EDP en las figuras 23 y 25 y la influencia del consumo en los valores del medidor EDP, ya que esta el momento sólo se habían expuesto hasta el momento los tiempos de ejecución.

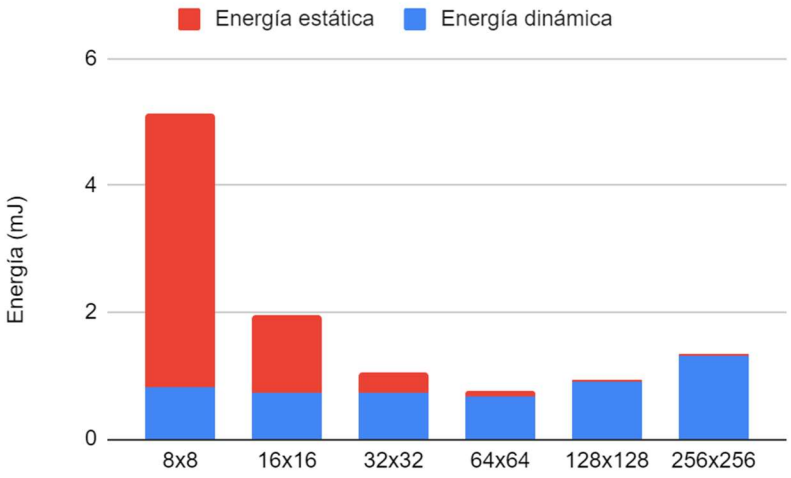

Figura 40: Consumo energético para W1 con configuraciones de array uniformes

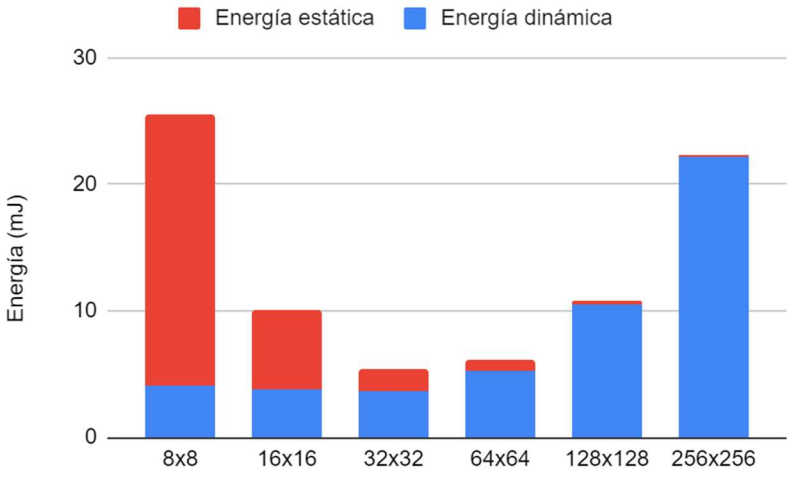

Figura 41: Consumo energético para W2 con configuraciones de array uniformes

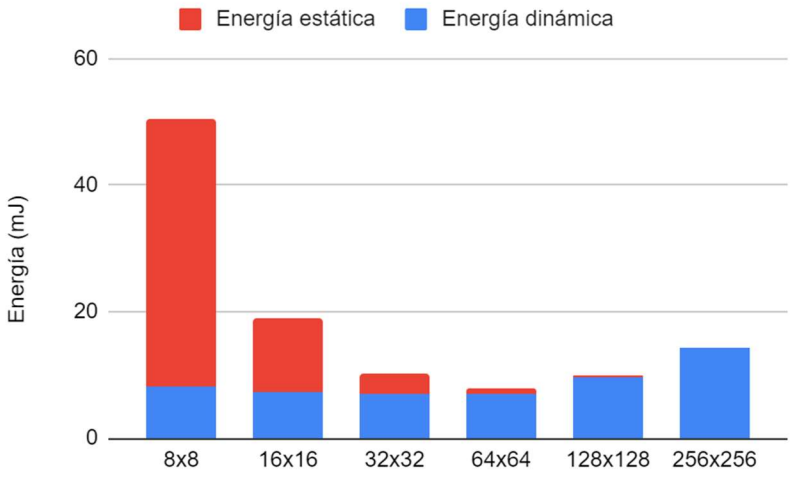

Figura 42:Consumo energético para W3 con configuraciones de array uniformes

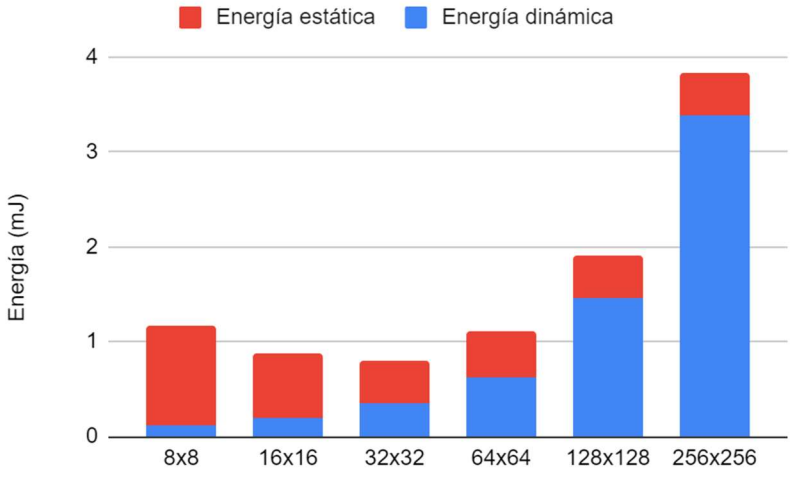

Figura 43: Consumo energético para W4 con configuraciones de array uniformes

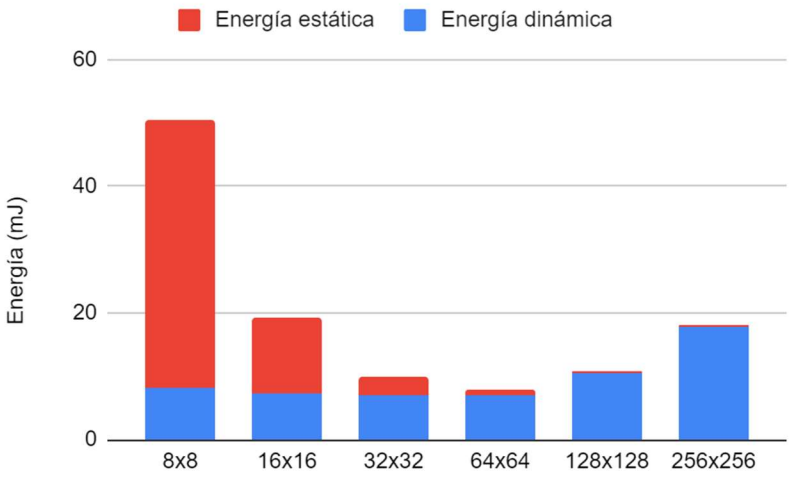

Figura 44: Consumo energético para W5 con configuraciones de array uniformes

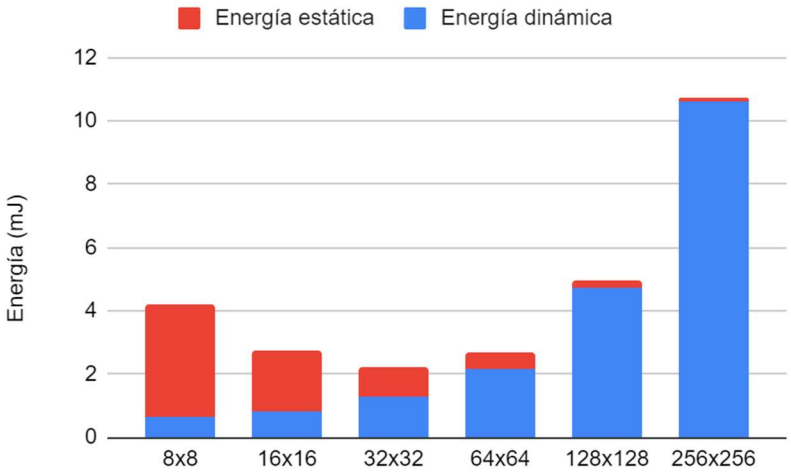

Figura 45:Consumo energético para W6 con configuraciones de array uniformes

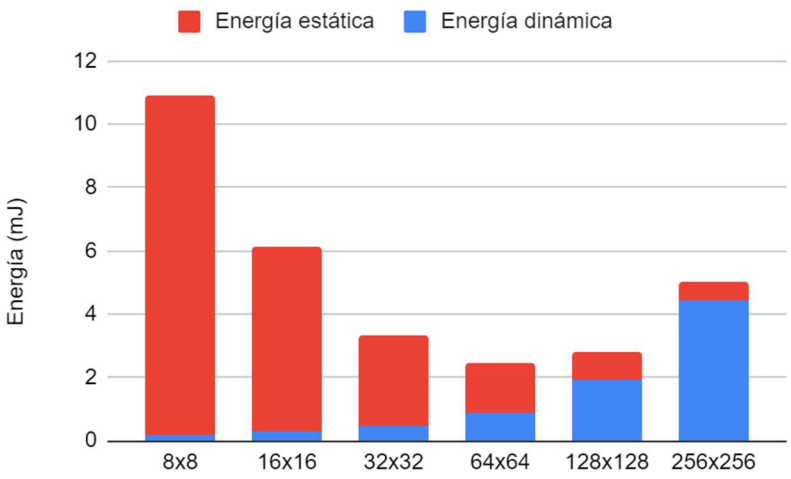

Figura 46: Consumo energético para W7 con configuraciones de array uniformes

Se pueden realizar dos observaciones de las gráficas respecto al consumo energético.

La primera es que a medida que aumenta el tamaño del array, para todas las cargas, se produce una inversión progresiva en los consumos, para arquitecturas pequeñas el consumo estático supone la mayor parte del consumo total mientras que en 256 x 256 el consumo dinámico supone la mayor parte del consumo.

Esto se debe a que conforme se aumenta el tamaño, el tiempo de ejecución se reduce considerablemente, tal y como ya se ha visto en la *figura 22,* produciendo por tanto disminuciones proporcionales en el consumo estático y que éste depende directamente del tiempo de ejecución.

Por otro lado, la segunda observación es respecto al consumo dinámico y total. Para todas las cargas empleadas en el experimento el consumo dinámico se mantiene estable y empieza a aumentar considerablemente a partir del tamaño 64x64.

A su vez y directamente ocasionado por los dos comportamientos descritos al respecto a la energía estática y dinámica, el consumo total para todas las cargas disminuye desde las configuraciones 8x8 hasta llegar a las configuraciones 32x32 y 64x64, dependiendo de la carga. A partir de estos tamaños el consumo total vuelve a incrementar con cada aumento del tamaño.

A continuación, se muestra lo que sucede entre esas dos configuraciones, y cómo varía el consumo en disposiciones irregulares dentro de esos rangos de tamaño.

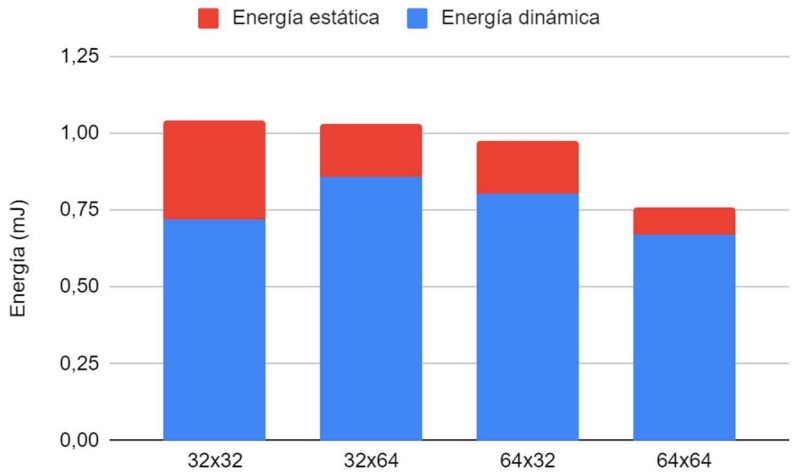

Figura 47: Consumo energético para W1 con configuraciones de array irregulares

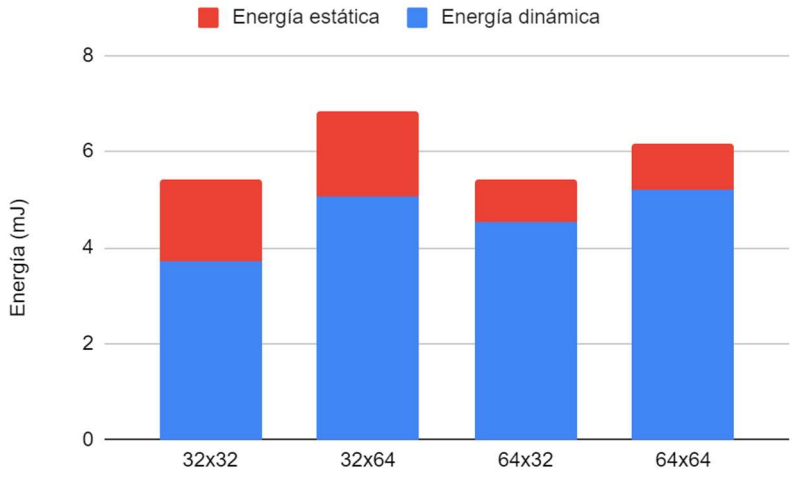

Figura 48: Consumo energético para W2 con configuraciones de array irregulares

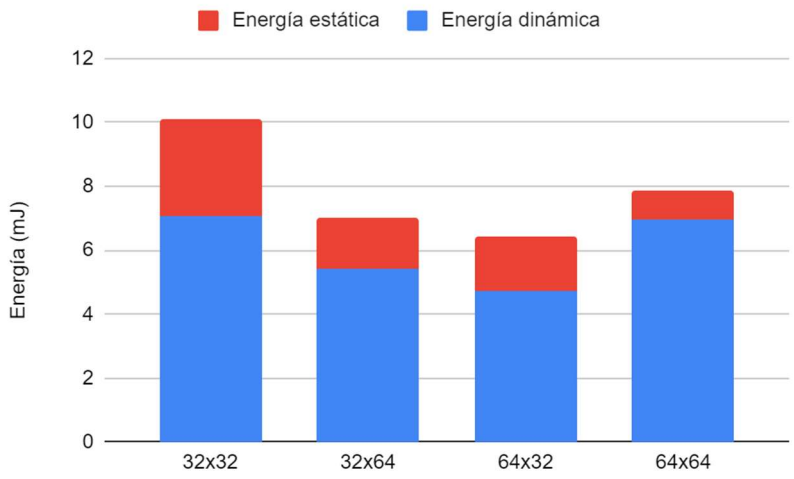

Figura 49: Consumo energético para W3 con configuraciones de array irregulares

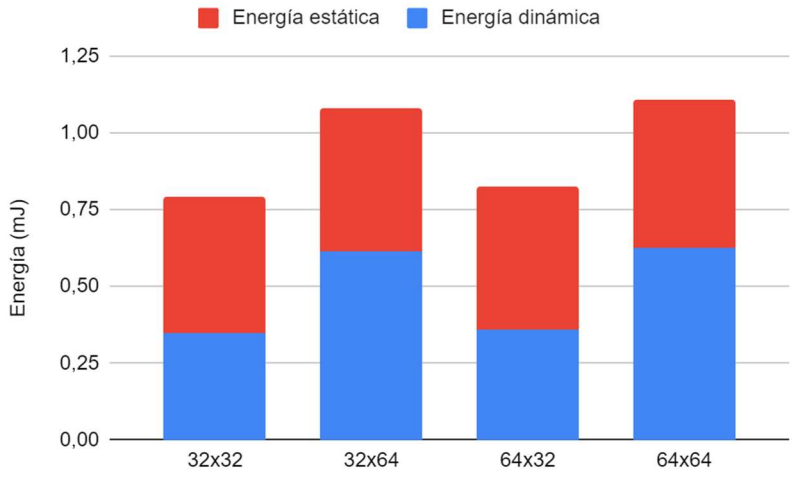

Figura 50:Consumo energético para W4 con configuraciones de array irregulares

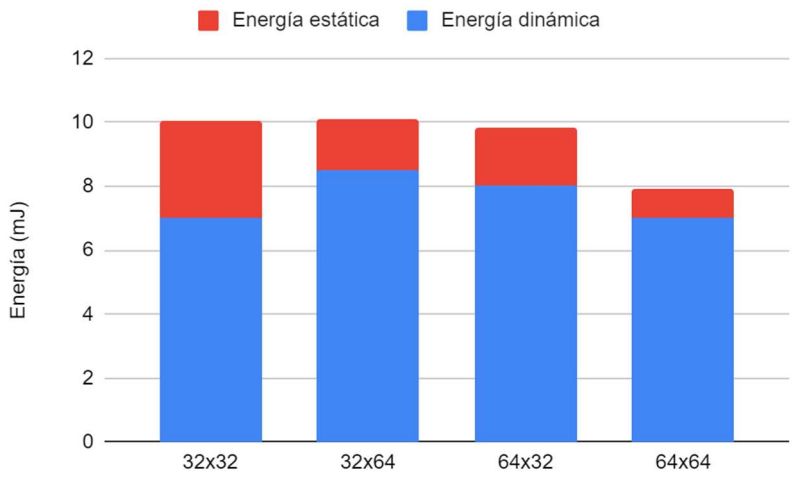

Figura 51: Consumo energético para W5 con configuraciones de array irregulares

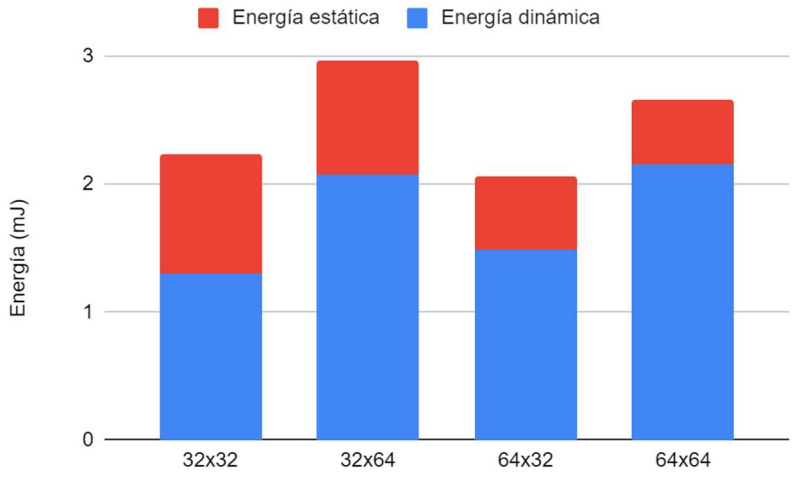

Figura 52: Consumo energético para W6 con configuraciones de array irregulares

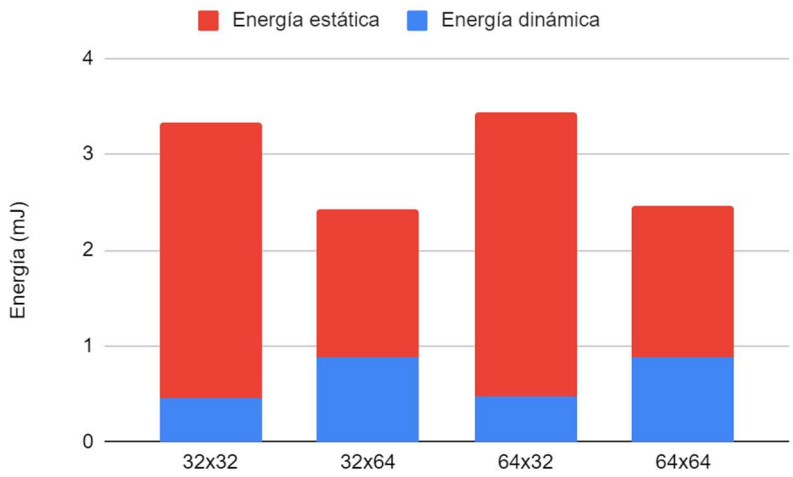

Figura 53: Consumo energético para W7 con configuraciones de array irregulars

En estos gráficos se observa cómo las cargas W1, W3 y W5 no presentan grandes cambios en el consumo energético en las configuraciones irregulares al igual que tampoco presentaban poca sensibilidad a estas configuraciones en el porcentaje de utilización como ya se vio en la *sección 6.3*.

Por el contrario, en las cargas W2, W4, W6 y W7 se producen diferencias notables entre las configuraciones irregulares de la siguiente manera: Para la configuración óptima de cada una de ellas ya estudiada, se reduce el consumo estático debido a la disminución del tiempo de ejecución y además también se reduce el consumo dinámico de forma más notables en algunas cargas como en W7 o W6 y de forma más modesta en la carga W2. Para la carga W4 como se ha estudiado en secciones anteriores no resulta interesante superar el tamaño 32x32, pero a pesar de ello en la configuración 64x32 reporta un menor consumo que la 32x64 o la 64x64 y esto podría ser útil en el caso que hubiera que llegar a un compromiso para mejorar la ejecución de más cargas y no sólo las que se parezcan a esta, y por tanto hubiese que aumentar el número de PE.

Otro hecho digno de mención es que este último fenómeno descrito se puede extender a todas las cargas del proyecto. Es decir, todas las cargas presentan un menor consumo dinámico en 32x64 que en 64x32 en mayor o en menor medida. Esto da más sentido al hecho de que en la *figura 25*, el medidor EDP presentase una mayor concentración de puntos cercanos al 0,5 que 32x64.

Este fenómeno relativo al consumo dinámico en las configuraciones 64x32 produce que, en todas las cargas a excepción de w7, el consumo total disminuya o se mantenga respecto a 32x64 y por tanto obtengan mejor EDP tal y como refleja la gráfica asociada.

6.5 Anchos de banda mínimos requeridos

Después de analizar todos los parámetros anteriores respecto a los experimentos realizados, queda por ver si los resultados obtenidos en las configuraciones empleadas son realistas. Como ya se había indicado el simulador SCALE-Sim simula ejecuciones ideales sin limitaciones hardware y por lo tanto las ejecuciones simuladas no realizan ciclos de parada. En esta sección analizaremos si en la realidad los resultados de los experimentos serían alcanzables o si por el contrario se pueden considerar meramente teóricos. Esto lo decidiremos mediante el estudio de uno de los parámetros más limitantes en la tecnología actual, el ancho de banda de la memoria principal del sistema.

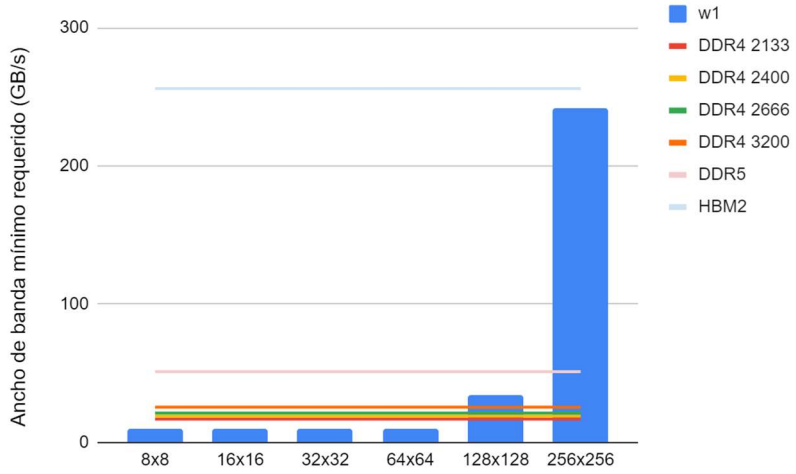

Figura 54: Ancho de banda requerido para ejecuciones cuadradas en W1

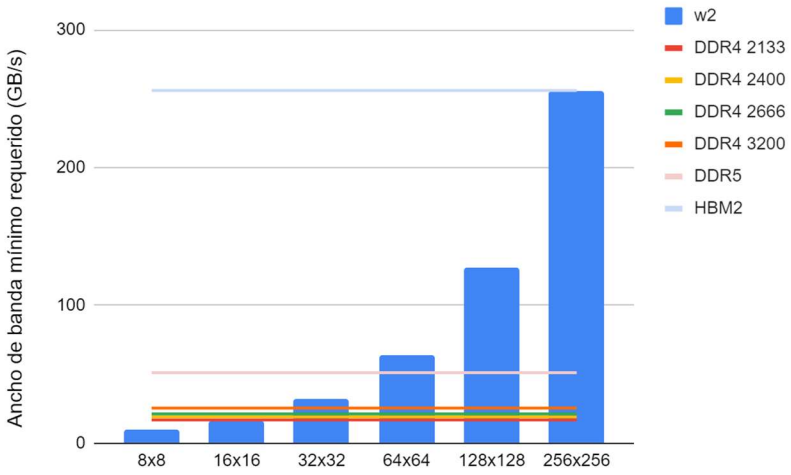

Figura 55: Ancho de banda requerido para ejecuciones cuadradas en W2

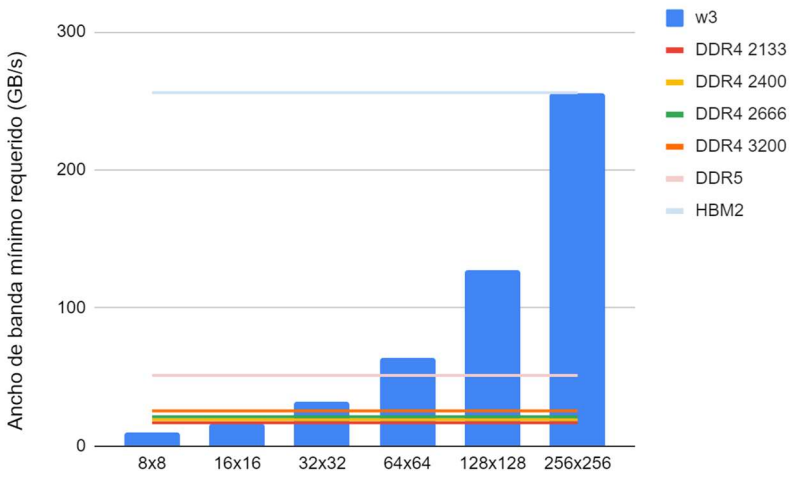

Figura 56: Ancho de banda requerido para ejecuciones cuadradas en W3

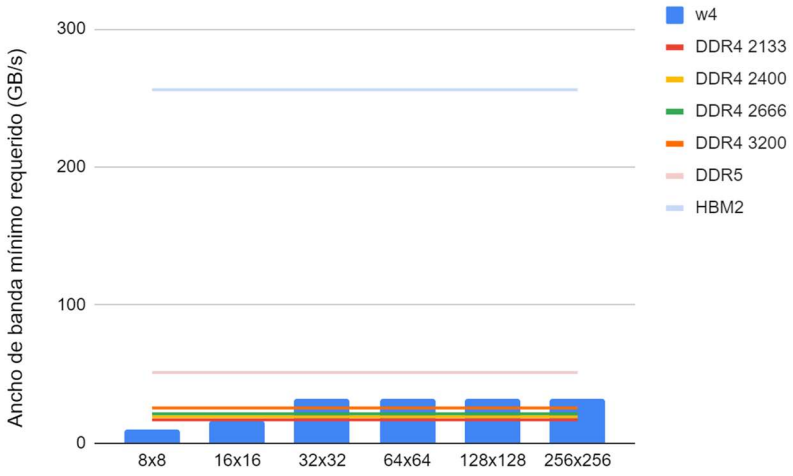

Figura 57: Ancho de banda requerido para ejecuciones cuadradas en W4

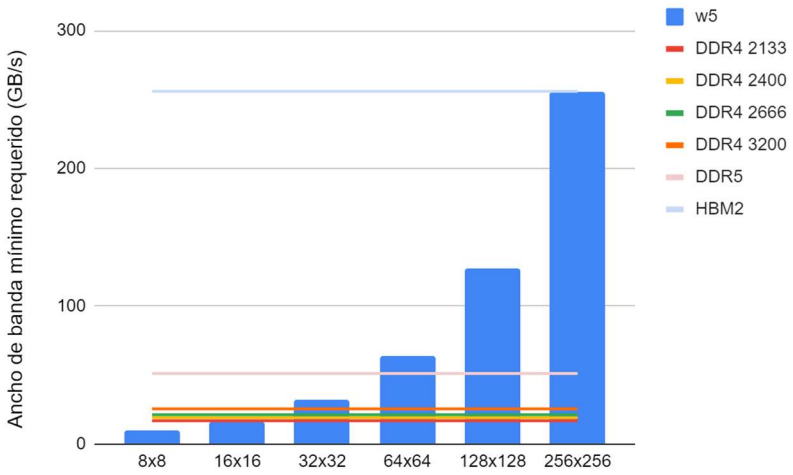

Figura 58: Ancho de banda requerido para ejecuciones cuadradas en W5

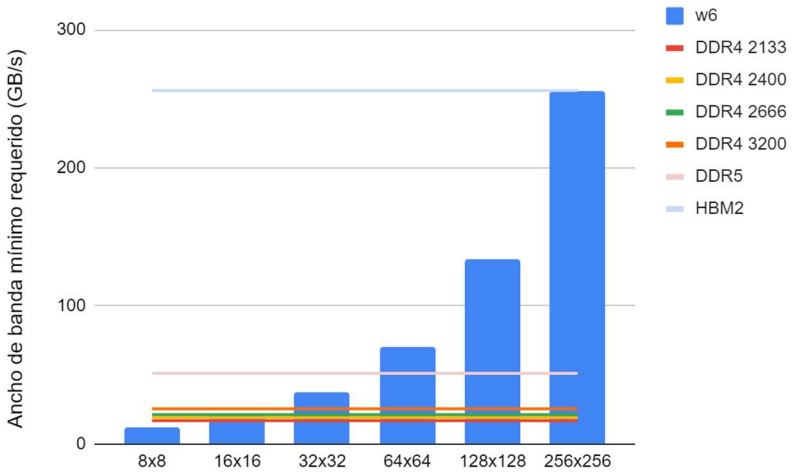

Figura 59: Ancho de banda requerido para ejecuciones cuadradas en W6

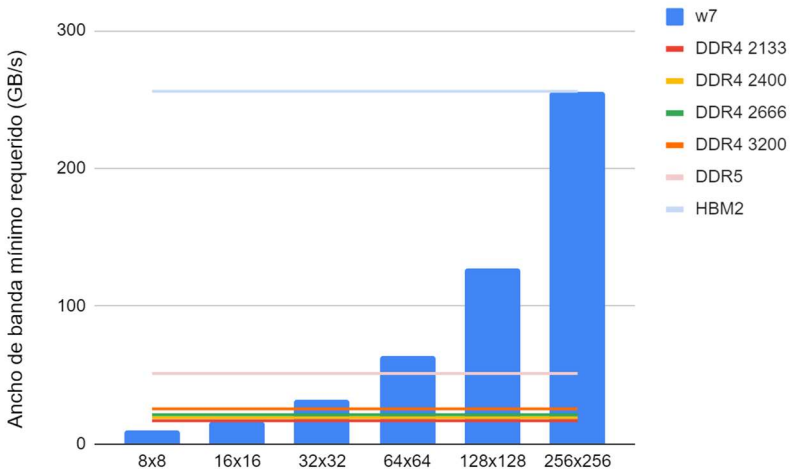

Figura 60: Ancho de banda requerido para ejecuciones cuadradas en W7

Los anchos de banda resultantes tienen un comportamiento exponencial conforme aumenta el tamaño del array. El ancho de banda requerido se dispara a partir de 64x64 en la mayoría de las cargas salvo en W1 que se dispara en 256x256 y en W4 por su caso especial que ya se ha tratado numerosas veces en este documento. En las gráficas también se pueden ver barras horizontales que representan el ancho de banda máximo que soportan memorias de distintas tecnologías actuales, esto facilita ver qué configuraciones son realistas y qué configuraciones no lo son. A efectos prácticos, todas las configuraciones de todas las cargas podrían ser cubiertas con una memoria HBM2, pero ya hemos visto anteriormente que configuraciones superiores a 64x64 no son interesantes.

Las configuraciones desde 8x8 hasta 16x16 son viables utilizando memorias DDR4 con distintas frecuencias, pero nuevamente este proyecto ha demostrado que estas configuraciones no son óptimas y por tanto no resultan de interés.

Las configuraciones más interesantes se encuentran entre 32x32 y 64x64.

Una memoria DDR5 es capaz de cubrir las demandas de ancho de banda en las configuraciones 32x32 para todas las cargas, mientras que sería necesaria una memoria HBM para que la configuración 64x64 alcance valores cercanos en la realidad a los valores que aporta el simulador.

6.6 Impacto de limitaciones HW en prestaciones

El simulador SCALE-Sim no contempla limitaciones hardware en las simulaciones tal y como ya se ha expuesto con anterioridad. En esta sección se analiza el impacto en el tiempo de ejecución que tendría introducir limitaciones hardware de memoria principal. Concretamente se estudia el impacto de las limitaciones de ancho de banda que supondría implementar distintas tecnologías para la memoria principal del sistema.

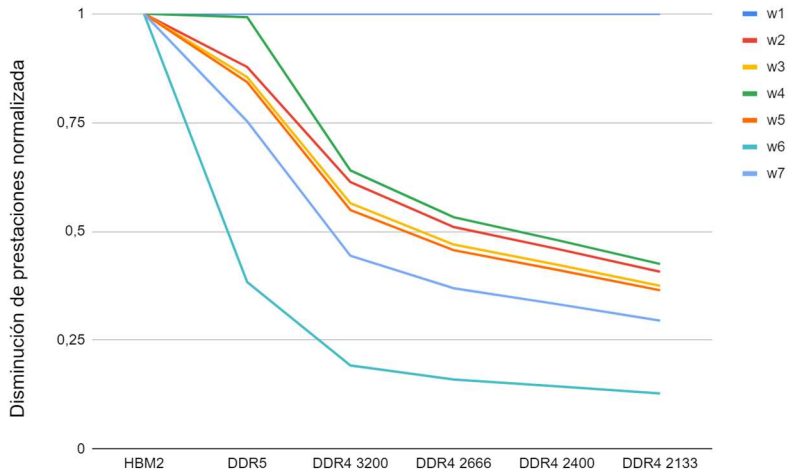

Figura 61: Impacto de limitaciones HW de memoria en 64x64

En la gráfica se puede observar la caída en prestaciones del tiempo de ejecución para las distintas cargas en la configuración 64x64. Los tiempos de ejecución están normalizados respecto al tiempo de ejecución ideal que ofrece el simulador (sin ciclos de parada). De este modo el valor "1" significa que se alcanza un tiempo de ejecución ideal. Conforme más se alejen los puntos de este valor más lejos están del tiempo de ejecución ideal, por ejemplo, un punto cercano al 0,5 significa que las prestaciones ofrecidas con esa configuración de memoria caen hasta la mitad respecto al ideal (duplicando el tiempo de ejecución).

Para la obtención del aumento en los ciclos de ejecución introducido por las distintas tecnologías de memoria, se ha hecho una estimación respecto al peor de los casos posibles durante la ejecución de cada capa. Esto se ha realizado sumando los anchos de banda mínimos requeridos para lectura de IFMAP, FILTER y escritura de OFMAP en la memoria principal durante la ejecución de cada capa y dividiendo el ancho de banda resultante entre el ancho de banda máximo que ofrece cada tecnología de memoria.

Después se multiplican los ciclos de ejecución de cada capa por el valor anteriormente mencionado, obteniendo el nuevo recuento de ciclos de ejecución. Finalmente se suman los ciclos de todas las capas obteniendo el tiempo de ejecución total de la carga. Para acabar se divide el valor de los ciclos de ejecución ideales con el número de ciclos de ejecución obtenidos y se obtiene la caída en prestaciones respecto al tiempo ideal ofrecido por la simulación.

Volviendo a la *figura 61,* se ve cómo 64x64 se podría evitar el coste de implementar la memoria principal con una HBM2 (de acuerdo con los anchos de banda mínimos requeridos expuestos la *sección 6.5*), sustituyendo ésta por una memoria DDR5 con caídas en prestaciones, en 6 de las 7 cargas empleadas, inferiores al 20%. Para implementaciones de memorias con tecnología que ofrezcan anchos de banda menores cómo DDR4 3200 las prestaciones caerían a valores dentro del rango de 60% a un 20% del valor ideal.

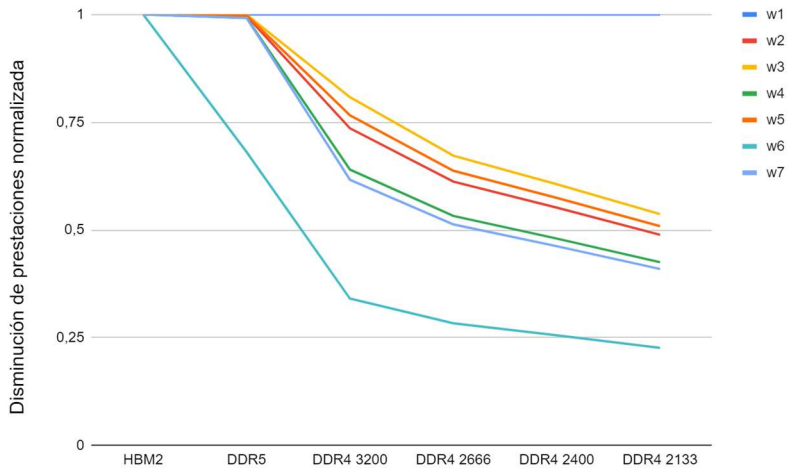

Figura 62: Impacto de limitaciones HW de memoria en 32x32

En la configuración 32x32 encontramos un comportamiento similar al de la gráfica con una configuración 64x64. En este caso tal y como se apreciaba en las gráficas de la sección anterior, para todas las cargas en la configuración 32x32 el ancho de banda mínimo requerido era cubierto por una DDR5. Aquí a pesar de tratarse de un estudio realizado sobre el contexto de que se den todos los peores casos posibles durante la ejecución, una DDR5 sigue cubriendo casi a la perfección los anchos de banda requeridos para todas las cargas (aunque se observa una disminución en prestaciones prácticamente imperceptible).

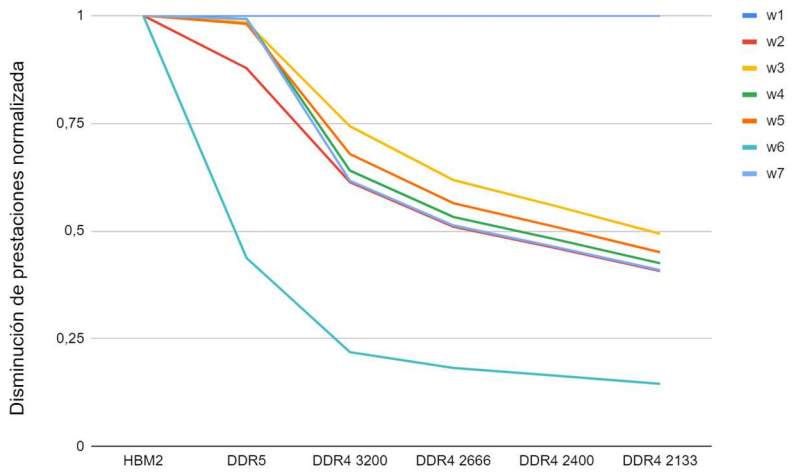

Figura 63: Impacto de limitaciones HW de memoria en 32x64

Finalmente, la configuración 32x64 presenta caídas en el rendimiento sensiblemente menores que la configuración 64x64 si se implementa la memoria principal con una tecnología DDR5 tal y como se puede observar comparando las *figuras 63* y *61*. Si a este hecho se le añaden los resultados obtenidos sobre la EDP en la *figura 25* de la *sección 6.2* y las conclusiones del mismo apartado y de la *sección 6.4* respecto a esta configuración de 32x64 hacen que posiblemente se pueda considerar como la más óptima ya que ofrece valore competitivos en comparación con los que ofrece 64x64 en cuanto a valores de la EDP con la mitad de elementos de procesamiento y ahora estamos demostrando que además puede ser implementada con una tecnología DDR5 con un impacto menor en las prestaciones.

Comparando 32x64 con 32x32 a pesar de lo expuesto con anterioridad referente a que las prestaciones varían de forma imperceptible al implementarse con DDR5, hay que recordar que las prestaciones mostradas en la *figura 25* siguen postulando la configuración 32x32 como mejor.

Esta sección persigue analizar la viabilidad de implementar la memoria DRAM más barata posible con el mínimo impacto negativo en prestaciones para las configuraciones más interesantes según los resultados del proyecto (32x32, 32x64 y 64x64) y esto desemboca en la memoria DDR5. A pesar de esto esta sección también tiene un valor interesante si se desea observar el impacto que tendrían tecnologías de memoria con menor ancho de banda como DDR4 de distintas frecuencias echando un simple vistazo a las tres gráficas.

7 CAPÍTULO: Conclusiones

El presente trabajo es el primer TFG (en conjunto con el de Pau Castelló Ferrer) que se realiza sobre arrays sistólicos en el seno del grupo de investigación GAP. Con este trabajo se pretenden sentar las bases y los conocimientos para el desarrollo de la investigación en esta línea.

En este documento se ha presentado un análisis sobre el estado del arte de los arrays sistólicos, flujos de datos y redes CNN. Además, se ha expuesto en detalle el funcionamiento del simulador SCALE-Sim con el que se ha realizado el presente proyecto. Esto nos ha permitido realizar un estudio en profundidad de los parámetros arquitectónicos de los arrays sistólicos

Hemos podido comprobar que la arquitectura de un array sistólico tiene multitud de parámetros referentes a la arquitectura hardware y que la computación de redes neuronales no es algo trivial. Gracias al simulador hemos podido analizar el impacto en las prestaciones y el consumo del dimensionamiento del array mediante los distintos experimentos presentados en este Trabajo Final de Grado sobre un conjunto de cargas representativas.

Tras realizar los experimentos hemos podido extraer las conclusiones siguientes:

1. Para las cargas consideradas y el flujo OS sin precarga, las configuraciones 64x64 y 64x32 desde el punto de vista del coste en área de silicio, consumo energético y tiempos de ejecución son las mejores (si se cubren los anchos de banda requeridos) y por tanto para un espectro variado de redes CNN.
2. Las arquitecturas cuadradas o con distribución uniforme permiten alcanzar prestaciones aceptables para un mayor número de redes CNN, mientras que arquitecturas irregulares o rectangulares permiten alcanzar las mejores prestaciones, aunque para un número menor de redes CNN.
3. Durante la fase de diseño de un acelerador puede ser de utilidad emplear el simulador SCALE-Sim para hacer un análisis de los tipos de cargas que más frecuentemente se ejecutarán en el acelerador en desarrollo y con ello optimizar el dimensionamiento del array para esas cargas.
4. Es vital asegurar que se cumplen los anchos de banda mínimos requeridos que marca el simulador o bien que se suministran valores cercanos a estos para obtener las mejores prestaciones.
5. La configuración 32x32 es capaz, incluso cuando se cumplen los peores casos posibles durante la ejecución respecto a anchos de banda, de ofrecer prestaciones cercanas a las que se muestran en las simulaciones con una memoria DDR5, mientras que las

configuraciones superiores cómo 64x32 y 64x64 requieren de una memoria HBM. También se ha medido en la *sección 6.6* el impacto en las prestaciones que supondría emplear memorias que proporcionan un ancho de banda menor al requerido según los resultados expuestos en la *sección 6.5* resultando en la mayoría de los casos una pérdida de prestaciones que podrían resultar interesantes según los requerimientos de las partes interesadas.

6. Configuraciones optimizadas de *scratchpad* posiblemente consigan que el ancho de banda requerido por las cargas para la configuración 64x64 pueda ser cubierto por una DDR5 obteniendo las máximas prestaciones y, por tanto, valores muy cercanos a los ofrecidos por el simulador. A pesar de esto hay que recordar que debido a la extensión del proyecto no se han realizado experimentos con variaciones en el tamaño de *scratchpad*.

Como trabajo futuro, se pretende preparar una publicación científica internacional basada en la ampliación de algunos de los principales resultados obtenidos en este TFG. En concreto, se hará especial énfasis en la parte de consumo energético y el impacto de distintas tecnologías DRAM sobre el tiempo de ejecución.

7.1 Competencias transversales del proyecto

Durante el trabajo realizado en este Trabajo Fin de Grado se han desarrollado las siguientes competencias trasversales de la UPV:

CT-01. Comprensión e integración: Se ha recuperado información de diversos estudios científicos sobre arrays sistólicos que ha sido redescrita a partir de la comprensión del alumno. Además, los conocimientos aprendidos han sido empleados en este trabajo.

CT-02. Aplicación y pensamiento práctico: Durante el desarrollo del proyecto se han tomado decisiones que implican actuar, más allá de los conocimientos puramente teóricos. Se ha tomado la decisión de desarrollar dos scripts en Bash para agilizar la fase de experimentación. También se ha tomado la decisión de modificar archivos sobre la topología de las cargas que ofrecían originalmente modeladas los desarrolladores del simulador con el propósito de arreglar un problema existente e inesperado. (*CAPÍTULO 6, sección 6.1*)

CT-13. Instrumental específica: Se han identificado las herramientas adecuadas para la resolución de un problema específico y se han integrado con éxito aportando una solución. Esto está relacionado con lo descrito en la competencia anterior, se ha explotado el scripting en Bash para automatizar el proceso de lanzamiento de ejecuciones y tratamiento de los datos obtenidos.

CT-07. Responsabilidad ética, medioambiental y profesional: Esta competencia calificada por la UPV de bidimensional, se ha desarrollado en este trabajo en la dimensión de la responsabilidad ética y profesional al tratar de referenciar correctamente en este documento las fuentes de trabajos y estudios previos ya existentes, intentando evitar plagios y respetando la propiedad intelectual de terceros. Por otro lado, también se ha desarrollado, aunque en menor medida, la dimensión medioambiental de esta competencia ya que en este trabajo se ha tenido muy en cuenta el consumo energético de los arrays sistólicos simulados. Una muestra de ello es que la EDP ha sido una variable decisiva a la hora de elegir las configuraciones más interesantes.

8 CAPÍTULO: Bibliografía

[1]"Red neuronal artificial", Es.wikipedia.org, 2019. [Online]. Disponible: https://es.wikipedia.org/wiki/Red_neuronal_artificial. [Recuperado: 18- Jun- 2019].

[2]"¿Qué son y cuáles son los tipos de redes neuronales? | Neuronoticias", Neuronoticias, 2019. [Online]. Disponible: http://neuronoticias.com/2018-06-27/tipos-de-redes-neuronales/. [Recuperado: 19- Jun- 2019].

[3]"Part 1: All about Artificial Neural Networks - Intuition", Linkedin.com, 2019. [Online]. Disponible: https://www.linkedin.com/pulse/part-1-all-artificial-neural-networks-intuition-amsal-gilani. [Recuperado: 20- Jun- 2019].

[4]"The Essence of Artificial Neural Networks", *Medium*, 2019. [Online]. Disponible: https://medium.com/@ivanliljeqvist/the-essence-of-artificial-neural-networks-5de300c995d6. [Recuperado: 23- Jun- 2019].

[5]"Clasificación de las Redes Neuronales Artificiales.", *Redes-neuronales.com.es*, 2019. [Online]. Disponible: http://www.redes-neuronales.com.es/tutorial-redes-neuronales/clasificacion-de-las-redes-neuronales-artificiales.htm. [Recuperado: 20- Jun- 2019].

[6]"Construye tu primer clasificador de Deep Learning con TensorFlow: Ejemplo de razas de perros", *Medium*, 2019. [Online]. Disponible: https://medium.com/datos-y-ciencia/construye-tu-primer-clasificador-de-deep-learning-con-tensorflow-ejemplo-de-razas-de-perros-ed218bb4df89. [Recuperado: 1- Jul- 2019].

[7]"Neural Network Models in R", *DataCamp Community*, 2019. [Online]. Disponible: https://www.datacamp.com/community/tutorials/neural-network-models-r. [Recuperado: 09- Jul- 2019].

[8]"Statistics is Freaking Hard: WTF is Activation function", *Medium*, 2019. [Online]. Disponible: https://towardsdatascience.com/statistics-is-freaking-hard-wtf-is-activation-function-df8342cdf292. [Recuperado: 11- Jul- 2019].

[9]"An in-depth look at Google's first Tensor Processing Unit (TPU) | Google Cloud Blog", *Google Cloud Blog*, 2019. [Online]. Disponible:

https://cloud.google.com/blog/products/gcp/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu. [Recuperado: 21- Jul- 2019].

[10] Y.-H. Chen, T. Krishna, J. Emer, y V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” en International Solid-State Circuits Conference, ser. ISSCC, 2016.

[11] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina y T. Krishna, *SCALE-Sim: Systolic CNN Accelerator Simulator* en Nueva York: Cornell University, 2019.

[12]"Arm Chooses NVIDIA Open-Source CNN AI Chip Technology", *Forbes.com*, 2019. [Online]. Disponible: https://www.forbes.com/sites/moorinsights/2018/03/29/arm-chooses-nvidia-open-source-ai-chip-technology/#71f06fd1e509. [Recuperado: 21- Jul- 2019].

[13]"NVIDIA Deep Learning Accelerator", *Nvdla.org*, 2019. [Online]. Disponible: http://nvdla.org/. [Recuperado: 8- Ago- 2019].

[14]"ARM-software/SCALE-Sim", *GitHub*, 2019. [Online]. Disponible: https://github.com/ARM-software/SCALE-Sim. [Recuperado: 20- Ago- 2019].

9 ANEXOS

9.1 ANEXO 1

```
    #!/bin/bash
    i=$3
    j=$6
    c=$(( $5 + 1))
    z=$(( $8 + 1))
    if [ "$2" = "all" ]; then dat=(os ws is)
    else
          dat=($2)
    fi
    if [ "$1" == "--help" ]; then
            echo "PARAMETERS:"
          echo "\$1: [-s for squared only, by default is not sqared only, -r for non square and -t for only non
square]"
          echo "\$2: [dataFlow(os/rw/is/all)]"
          echo "\$3: [height_ini]"
          echo "\$4: [mult_fact_h]"
          echo "\$5: [h_max]"
          echo "\$6: [widh_ini]"
          echo "\$7: [mult_fact_w]"
          echo "\$8: [W_max]"
          echo "Last 3 parameters will be ignored in -s configuration, so is not necessary to set
[widh_ini],[mult_fact_w],[W_max] in that case."
    elif [ "$1" == "-s" ]; then
            for df in ${dat[*]}
            do
                    while [  $i -lt $c ];
                    do

                                    for w in $(seq 1 1 7)
                                    do
                                            (
                                            echo "[general]"
                                            echo "run_name = "\"w"$w"_"$i"x"$i"_"$df"\"
                                            echo
                                            echo "[architecture_presets]"
                                            echo "ArrayHeight:    "$i
                                            echo "ArrayWidth:     "$i
                                            echo "IfmapSramSz:    "512
                                            echo "FilterSramSz:   "512
                                            echo "OfmapSramSz:    "256
                                            echo "IfmapOffset:    "0
                                            echo "FilterOffset:   "10000000
                                            echo "OfmapOffset:    "20000000
                                            echo "Dataflow:       "$df
                                            ) >> ./configs/fig6cfgs/w"$w"_"$i"x"$i"_"$df".cfg
                                            python3 scale.py -
arch_config=configs/fig6cfgs/w"$w"_"$i"x"$i"_"$df".cfg -network=topologies/fig6ws/w"$w".csv
                                    done
                                    i=$(( $i * $4 ))
                            if [ "$3" == "$5" ]; then
                            i=$(( $i * $4 + 1 ))
                            fi
                    done
                i=$3
            done
```

```
    else
          for df in ${dat[*]}
          do
                  while [  $i -lt $c ];
                  do
                          while [  $j -lt $z ];
                          do
                                for w in $(seq 1 1 7)
                                do
                                       if [ $i -ne $j ]; then
                                              (
                                              echo "[general]"
                                              echo "run_name = "\"w"$w"_"$i"x"$j"_"$df"\"
                                              echo
                                              echo "[architecture_presets]"
                                              echo "ArrayHeight:    "$i
                                              echo "ArrayWidth:     "$j
                                              echo "IfmapSramSz:    "512
                                              echo "FilterSramSz:   "512
                                              echo "OfmapSramSz:    "256
                                              echo "IfmapOffset:    "0
                                              echo "FilterOffset:   "10000000
                                              echo "OfmapOffset:    "20000000
                                              echo "Dataflow:       "$df
                                              ) >> ./configs/fig6cfgsR/w"$w"_"$i"x"$j"_"$df".cfg
                                              python3 scale.py -
arch_config=configs/fig6cfgsR/w"$w"_"$i"x"$j"_"$df".cfg -network=topologies/fig6ws/w"$w".csv
                                       fi
                                done
                                if [ "$6" == "$8" ]; then
                                j=$(( $j * $7 + 1))
                                else
                                j=$(( $j * $7 ))
                                fi
                          done
                         j=$6
                         if [ "$3" == "$5" ]; then
                         i=$(( $i * $4 + 1 ))
                         else
                         i=$(( $i * $4 ))
                         fi
                  done
                 i=$3
          done
    fi
```

9.2 ANEXO 2

```
#!/bin/bash
path=outputs
ifmapR=0
filterR=0
ofmW=0
first=0
for dir in $(find $path -type d -name w* )
 do
       dirname="${dir##*/}"
       echo "Counting in $dirname"
       for file in $(find $dir -type f -name "*sram_read.csv")
        do
              filename="${file##*/}"
              while read line_read ; do
                     for line in $line_read; do
                            if [ $first -gt 0 ] && [ $line != "," ]
                             then
                                   line=${line%,}
                                   if [ $line -ge 10000000 ]
                                    then
                                          let filterR++
                                   else
                                          let ifmapR++
                                   fi
                            fi
                            first=1
                     done
              first=0
              done <$file
       echo "File $filename in $dirname computed with values ifmaps: $ifmapR
       filters: $filterR"
       done
       (
       echo "Output: $dirname"
       echo "Total SRAM_IFMAP reads during execution: $ifmapR"
       echo "Total SRAM_FILTER reads during execution: $filterR"
       ) >> ./output_access.out

       ifmapR=0
       filterR=0
       for file in $(find $dir -type f -name "*sram_write.csv")
        do
              filename="${file##*/}"
              while read line_read ; do
                     for line in $line_read; do
                            if [ $first -gt 0 ] && [ $line != "," ]
                             then
                                   let ofmW++
                            fi
                            first=1
                     done
              first=0
              done <$file
       echo "File $filename in $dirname computed with value ofmap: $ofmW"
       done
       (
       echo "Total SRAM_OFMAP write during execution: $ofmW"
       echo
       ) >> ./output_access.out
       ofmW=0

done
exit 0
```

-size (bytes) 524288

-block size (bytes) 8

-associativity 1

-exclusive read port 1

-exclusive write port 1

-UCA bank count 1

-technology (u) 0.032

-output/input bus width 64

-operating temperature (K) 350

-tag size (b) "default"

-access mode (normal, sequential, fast) - "fast"

-cache type (SRAM - only data array <ex:register files, buffers etc.>, SRAM_CACHE - tag & data array, DRAM_CACHE) - "SRAM"

-design objective (weight delay, dynamic power, leakage power, cycle time, area) 100:100:0:0:0

-deviate (delay, dynamic power, leakage power, cycle time, area) 20:100000:100000:100000:1000000

-NUCAdesign objective (weight delay, dynamic power, leakage power, cycle time, area) 100:100:0:0:0

-NUCAdeviate (delay, dynamic power, leakage power, cycle time, area) 10:10000:10000:10000:10000

-Optimize ED or ED^2 (ED, ED^2, NONE): "ED"

-Cache model (NUCA, UCA) - "UCA"

-NUCA bank count 0

-wire signalling (fullswing, lowswing, default) - "default"

-Print level (DETAILED, CONCISE) - "DETAILED"