Caracterización de mapeos multi-dataflow para modelos DNN dispersos en el acelerador Flexagon

Adrián Navarro[1], Francisco Muñoz-Martínez[1], José Cano[2], José L. Abellán[1]
Manuel E. Acacio[1]

Resumen— **La Multiplicación de Matrices Dispersas (SpMSpM) es el núcleo principal de muchas aplicaciones modernas, siendo eficazmente usada para la optimización del procesamiento de Redes Neuronales Profundas (DNNs). Esta operación viene caracterizada por su alta demanda de cómputo, memoria y energía. Es por ello que el diseño de arquitecturas especializadas para acelerar las operaciones SpMSpM se ha convertido en un tema candente. Estas arquitecturas pretenden obtener una mejor relación entre rendimiento y consumo energético mientras emplean menos silicio que los sistemas de cómputo tradicionales CPU y GPU. Concretamente, Flexagon ha sido recientemente publicado como el único acelerador flexible para SpMSpM que mediante un mismo sustrato hardware da soporte al flujo de datos óptimo a cada operación SpMSpM. En este trabajo se extiende el estudio realizado en Flexagon con las siguientes contribuciones: (1) consideramos y evaluamos una extensión a los flujos de datos originales de Flexagon basados en el producto transpuesto; (2) proponemos diversos mecanismos para realizar las Conversiones de formato Explícitas (ECs) que son necesarias para concatenar distintas SpMSpM que requieren formatos de compresión distintos en sus matrices (CSR o CSC); y (3) evaluamos el impacto que las ECs conllevan en la práctica, cuando se procesan todas las capas en modelos DNN dispersos. Nuestros resultados muestran que esta extensión al conjunto base de flujos de datos obtiene una mejora de rendimiento promedio del 51 % con respecto a los resultados originales, siendo esta mejora implementable en cualquier acelerador para SpMSpM sin aplicar cambios significativos. Además, nuestras propuestas permiten realizar ECs introduciendo tan solo entre un 1,0 % y un 1,5 % de sobrecarga con respecto al rendimiento ideal del acelerador.**

Palabras clave— **Acelerador Hardware, Redes Neuronales Profundas, Multiplicación de Matrices Dispersas, *Dataflows*, Formatos de Representación *Sparse*.**

I. Introducción

LAS matrices dispersas o *sparse*, que se caracterizan porque la mayoría de sus elementos (por encima del 99 % en algunos casos) tienen valor cero, están presentes en multitud de aplicaciones científicas de distintos tipos: álgebra lineal [1], análisis avanzado de grafos [2], simulación de dinámica molecular [3], *machine learning* [4], etc. La mayoría de estas aplicaciones trabajan con millones de parámetros y demandan mucho cómputo, memoria y consumo energético. Una de las operaciones más comunes en este tipo de aplicaciones son las Multiplicaciones de Matrices Dispersas o *Sparse Matrix-Sparse Matrix Multiplication* (SpMSpM), llegando a consumir más del 70 % de su cómputo total [5]

La forma más habitual de aprovechar los altos ratios de *sparsity* es a través del uso de formatos de representación comprimidos, los cuales solo almacenan los valores útiles (i.e., valores distintos de cero). Existen múltiples formatos, pero los más extendidos son CSR y CSC, que son complementarios entre sí [6]. Esto conlleva una notable reducción tanto del uso de memoria como del número de operaciones necesarias, lo que a su vez se traduce en un importante ahorro de cómputo y energía [7].

El impacto de las operaciones SpMSpM las ha convertido en un objetivo de optimización, lo que ha llevado al desarrollo de aceleradores específicos que permiten obtener el máximo rendimiento por vatio (e.g., [5, 8–10]). Para el procesamiento eficiente de las operaciones SpMSpM, cada uno de estos aceleradores implementa un flujo de datos o *dataflow* concreto, los cuales han sido clasificados en tres grandes grupos: *Inner-Product* (IP), *Outer-Product* (OP) y *Gustavson's* (Gust).

Recientemente se ha observado que para una misma operación SpMSpM cada *dataflow* puede obtener resultados muy distintos en función de los patrones de acceso a los datos, el reuso de datos y la estructura del *sparsity* de las matrices involucradas. Esto ha impulsado el desarrollo de aceleradores capaces de soportar múltiples *dataflows* bajo el mismo sustrato hardware a través de su reconfiguración (e.g., [7,11]).

Este trabajo se centra en la arquitectura Flexagon [7], el primer acelerador flexible y homogéneo que tiene como objetivo las Redes Neuronales Profundas o *Deep Neural Network*s (DNNs) como su carga de trabajo principal. Las cargas de DNN se diferencian de las científicas porque utilizan matrices mucho más pequeñas y tienen ratios de *sparsity* menores, lo cual es aprovechado por Flexagon para capturar sus patrones de acceso a memoria. Flexagon soporta los tres *dataflows* mencionados anteriormente y, además, propone una extensión a este conjunto aprovechando la complementariedad de los formatos CSR/CSC. Sin embargo, esta extensión no se evaluó en su propuesta original.

El soporte de todas las variantes de *dataflows* en Flexagon da lugar a que se requieran conversiones de formato para poder ejecutar de manera consecutiva

[1]Dpto. de Ingeniería y Tecnología de Computadores, Universidad de Murcia, e-mail: {adrian.fenollarn,francisco.munoz2,jlabellan,meacacio}@um.es.

[2]School of Computing Science, University of Glasgow, e-mail: Jose.CanoReyes@glasgow.ac.uk.

múltiples operaciones SpMSpM dependientes entre sí (e.g., procesamiento de dos capas consecutivas en una DNN mediante operaciones SpMSpM) si se quiere poder utilizar en cada una de estas operaciones el *dataflow* óptimo (ya que cada uno requiere que las matrices estén codificadas en formato CSR o CSC). La evaluación del impacto que tiene el costo de las conversiones de formato en el procesamiento de una DNN *sparse* es otro de los puntos que, aunque presentado, no se estima ni se proponen mecanismos para abordarlo en el trabajo original [7].

En resumen, los objetivos que se persigue alcanzar en este trabajo son:

- Cuantificar y evaluar exhaustivamente los beneficios de considerar la extensión a los *dataflows* soportados en Flexagon como una de sus ventajas clave. En este sentido, nuestros resultados muestran que considerar las seis variantes de *dataflows* da lugar a una mejora promedio del rendimiento de $1,51\times$ (hasta $1,92\times$) respecto a considerar únicamente las tres variantes originales para el mismo conjunto de evaluación.
- Proponer distintos mecanismos para dar soporte eficiente a las conversiones de formato en Flexagon. Hemos cuantificado, además, el impacto que estas conversiones suponen sobre el rendimiento global de la ejecución de modelos DNN completos, las cuales introducen una sobrecarga de hasta un $1,5\,\%$ respecto al rendimiento ideal.

El resto del documento se organiza del siguiente modo. La Sec. II presenta los fundamentos de las DNNs, los formatos de representación *sparse*, los *dataflows* para SpMSpM y la arquitectura Flexagon. La Sec. III introduce la metodología de evaluación y nuestra propuesta para realizar las conversiones de formato de manera eficiente en el acelerador. La Sec. IV presenta una caracterización de Flexagon considerando la extensión a sus flujos de datos. La Sec. V muestra una caracterización del impacto en el rendimiento global que tienen las conversiones de formato con nuestros mecanismos propuestos. Finalmente, la Sec. VI incluye las principales conclusiones del trabajo realizado.

II. Antecedentes y Trabajo Relacionado

A. Deep Neural Networks

Las Redes Neuronales Profundas o DNNs son ampliamente usadas en todo tipo de aplicaciones (e.g., procesamiento del lenguaje natural [4], segmentación de imágenes [12], generación de imágenes [13], etc.). Un modelo de DNN se compone de múltiples capas, las cuales tienen unos pesos o *weights* asociados y reciben entradas o *inputs* para realizar inferencias. El núcleo computacional principal de cada una de estas capas es la multiplicación de matrices, el cual consume hasta el $85-90\,\%$ del tiempo de ejecución total para la mayoría de modelos [14]. Para optimizar el procesamiento de las DNNs, la tendencia reciente es utilizar formatos de representación de matrices *sparse* aprovechando los altos ratios de *sparsity* resultantes tras la ejecución de sus funciones de activación y las técnicas de *pruning* que se les aplican [14], impulsando así el desarrollo de aceleradores hardware específicos para operaciones SpMSpM.

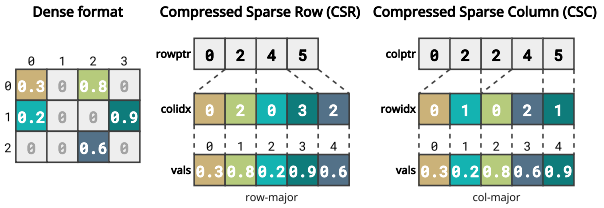


Fig. 1: Formatos de representación de matrices *sparse*. Nótese que CSR y CSC son complementarios.

B. Formatos de Representación de Matrices Sparse

Existe una gran variedad de formatos de representación para matrices *sparse* [6]. Este trabajo se centra en los formatos CSR y CSC, los cuales son complementarios. Se muestra un ejemplo en la Fig. 1. Una matriz almacenada en CSR utiliza tres vectores para almacenar los datos y los metadatos: `vals` contiene todos los valores distintos de cero de la matriz ordenados por filas; `colidx` contiene el índice de la columna de cada valor; y `rowptr` contiene los punteros (índices) al inicio de cada fila de la matriz. CSC emplea la misma estructura pero intercambiando las filas por columnas, tanto para datos como metadatos (i.e., `vals` está ordenado por columnas y se utilizan `rowidx` y `colptr` en su lugar).

Obsérvese que la ventaja clave de este formato es que dada una matriz A almacenada en formato CSR, su matriz traspuesta $A^{\intercal}$ se puede obtener con tan solo leer la matriz pero interpretándola como si estuviese representada en CSC y viceversa, evitando así la necesidad de realizar conversiones explícitas de formato en muchas situaciones. Esta homogeneidad permite a los aceleradores utilizar ambos formatos de representación utilizando la misma lógica de control.

Al igual que en trabajos anteriores (e.g., [7, 10]), utilizaremos el término fibra para referirnos a la fila/columna de una matriz en CSR/CSC que está siendo procesada en el acelerador para SpMSpM. Cada fibra contiene una lista de pares coordenada-valor ordenada por coordenada, denominados elementos.

C. Dataflows en aceleradores para SpMSpM

Siguiendo la taxonomía utilizada en Flexagon [7], una SpMSpM computa la operación $C_{M,N} = A_{M,K} \times B_{K,N}$, siendo los subíndices las dimensiones de cada matriz. En los aceleradores para SpMSpM se consideran, hasta la fecha, tres *dataflows* distintos: *Inner-Product* (IP), *Outer-Product* (OP) y *Gustavson's* (Gust) (e.g., [5], [8, 9] y [10, 15] implementan respectivamente cada uno de ellos; [7, 11] implementan varios simultáneamente).

Este conjunto puede ser entendido en función de qué matriz se considere como estacionaria o *stationary* (i.e., matriz que únicamente se lee una vez y se maximiza su reuso) y cual como transmitida o *streaming* (i.e., matriz que se debe leer desde memoria

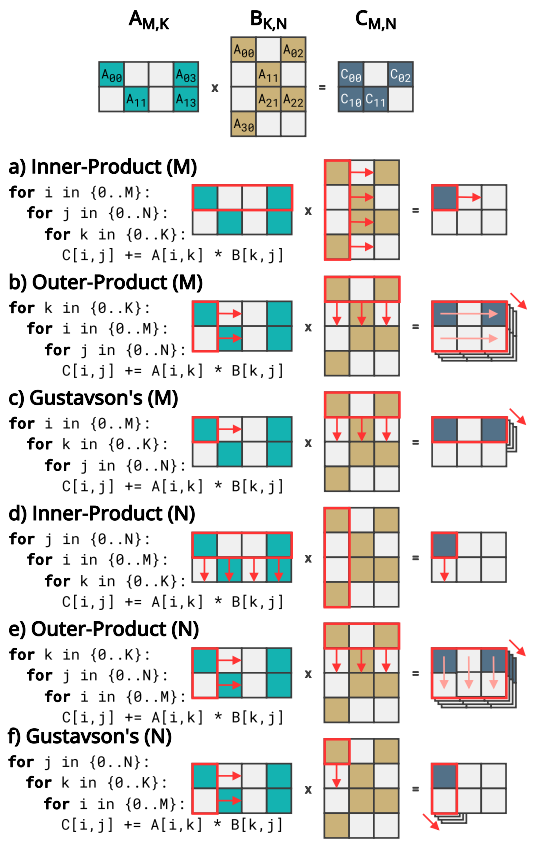


Fig. 2: *Dataflows* para SpMSpM en función del orden de sus bucles. Por simplicidad, se muestran matrices no comprimidas.

por cada iteración necesaria en la SpMSpM) dentro del acelerador. Así, siguiendo de nuevo la nomenclatura de Flexagon [7], definimos un total de seis *dataflows* distintos a partir de los anteriores, los cuales se muestran en la Fig. 2. Por un lado, tenemos los denominados M-$dataflows$ que consideran la coordenada M como estacionaria (i.e., en el bucle que compone la operación, la iteración sobre M se encuentra en un bucle más externo que N) y, por tanto, la matriz A es la *stationary* y B es la *streaming*. Por otro lado, en los N-$dataflows$ sucede el caso contrario y la matriz B es la *stationary* mientras A es la *streaming*.

C.1 Patrones de Reuso de *Dataflows* SpMSpM

Cada uno de estos *dataflows* exhibe patrones de acceso a memoria y reuso de datos distintos, lo cual afecta drásticamente al rendimiento dependiendo de la SpMSpM. La Tabla I resume el reuso de datos de cada matriz para cada uno de los seis *dataflows*. A continuación detallamos las características clave de cada uno de los tres principales tipos de *dataflows* (asumimos M-*dataflows* para la explicación):

- ***Inner-Product***. La reutilización de la matriz *streaming* es muy pobre ya que debe leerse completamente una vez por cada fibra de la matriz *stationary*. Sin embargo, todas las sumas parciales o *partial sums* (*psums*) de cada elemento de la matriz C se obtienen consecutivamente, por lo que cada *psum* puede reducirse directamente al valor $C_{i,j}$ completo y enviarse a memoria realizando una única escritura. Su mayor inconveniente son las intersecciones de índices dado que es necesario obtener las fibras $A_{i,K}$ y $B_{K,j}$ completas solo para comprobar qué índices k coinciden y producen una *psum*. Esto hace que IP no sea factible para matrices con alto grado de *sparsity*, pues el número de intersecciones efectivas es mucho menor que el tamaño total de las fibras [11].

Tabla I: Comparativa de las características clave de los seis *dataflows* para SpMSpM (adaptada de [7, 11]).

Dataflow	**Reuso de datos**			**Coste**	
	A	***B***	***C***	**Intersección de índices**	**Reducción de la salida**
IP-M	Alto	Bajo	Alto	$A_{i,K} \times B_{K,j}$ (Alto)	No requerido (Nulo)
OP-M	Alto	Alto	Bajo	$A_{M,k} \times B_{k,N}$ (Nulo)	Matriz $C_{M,N}$ (Alto)
Gust-M	Alto	Bajo	Medio	$A_{i,k} \times B_{k,N}$ (Bajo)	Fibra $C_{i,N}$ (Bajo)
IP-N	Bajo	Alto	Alto	$B_{K,j} \times A_{i,K}$ (Alto)	No requerido (Nulo)
OP-N	Alto	Alto	Bajo	$B_{k,N} \times A_{M,k}$ (Nulo)	Matriz $C_{N,M}$ (Alto)
Gust-N	Bajo	Alto	Medio	$B_{k,j} \times A_{M,k}$ (Bajo)	Fibra $C_{M,j}$ (Bajo)

- ***Outer-Product***. Consigue maximizar la reutilización de la matriz *streaming* al igual que de la matriz *stationary* (i.e., cada elemento de ambas matrices solo necesita ser obtenido una vez). Además, su esquema garantiza que todos los elementos de cada fibra $B_{k,N}$ siempre acabarán intersectando con los de las fibras $A_{M,k}$ (ya que comparten el mismo índice k), lo que lo hace muy eficiente. Por el contrario, la multiplicación de estas dos fibras producirá una matriz de *psums* de tamaño $C_{M,N}$, siendo necesario almacenarlas para realizar su reducción final posteriormente. De hecho, si una matriz de *psums* no puede almacenarse en la memoria *on-chip* del acelerador, los resultados tendrán que trasladarse a memoria *off-chip* y volver a leerse para ser reducidos posteriormente, lo que implica una gran cantidad de tráfico de datos, alta latencia y consumo de energía. Por tanto, OP presenta problemas de escalabilidad y tiene grandes requisitos de memoria.
- ***Gustavson's***. Ofrece una aproximación intermedia a IP y OP. Gust expone una baja reusabilidad de su matriz *streaming* al igual que IP, pero garantiza que para cada elemento $A_{i,j}$ siempre se producirán intersecciones con los elementos de la fibra $B_{k,N}$, evitando así sus costes como en OP. En este caso, se generarán fibras de *psums* que deberán reducirse posteriormente, requiriendo bastante menos memoria *on-chip* y permitiendo una mejor reutilización de la matriz C que OP. Esto hace que Gust sea especialmente adecuado para matrices pequeñas en las que todas las fibras parciales pueden almacenarse en la memoria *on-chip*.

D. *Flexagon*

Flexagon [7] es el primer acelerador hardware para SpMSpM completamente flexible y homogéneo capaz de poder ejecutar las seis variaciones de *dataflows* utilizando el mismo substrato hardware, permitiendo así utilizar aquel que mejor se ajuste a cada operación SpMSpM. Sus requisitos de área y consumo energético se diseñaron con las DNNs como objetivo principal, siendo así un acelerador específico para este tipo de cargas de trabajo.

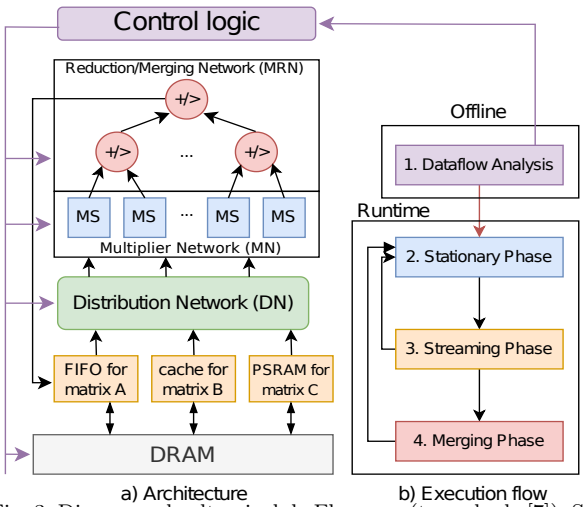


Fig. 3: Diagrama de alto nivel de Flexagon (tomado de [7]). Se considera la matriz A como *stationary* y B como *streaming*.

La Fig. 3 muestra un diagrama de alto nivel de la arquitectura de Flexagon y de su flujo de ejecución. Como se puede observar, el núcleo de Flexagon se compone de tres unidades principales. La Red de Distribución o *Distribution Network* (DN) se encarga de llevar los datos desde memoria hasta los multiplicadores de la Red de Multiplicadores o *Multiplier Network* (MN). Los valores de la matriz *stationary* se quedarán almacenados en un pequeño búfer dentro de cada multiplicador hasta que se hayan terminado de utilizar, mientras que los valores de la matriz *streaming* van fluyendo a través de ellos. Como resultado, cada multiplicación da lugar a una *psum* que será llevada a la Red de Fusión-Reducción o *Merger-Reduction Network* (MRN), que se encargará de reducir todas las *psums* en su valor final correspondiente y de enviarlo a memoria principal.

Además, Flexagon cuenta también con tres módulos SRAM *on-chip* específicos para cada una de las matrices involucradas en la operación SpMSpM. Por un lado, una memoria FIFO encola las lecturas únicas de la matriz *stationary*. Por otro lado, se utiliza una caché customizada para capturar los patrones de acceso heterogéneos de la matriz *streaming* y así mejorar su reuso. Por último, se utiliza una *psums* RAM (PSRAM) como unidad auxiliar durante la acumulación y reducción de resultados de la MRN.

D.1 *Dataflows* soportados en Flexagon

Flexagon da soporte a los seis *dataflows* a través de su reconfigurabilidad. La arquitectura fue diseñada para soportar `IP-M`, `OP-M` y `Gust-M`; pero se aprovechó el producto traspuesto de matrices ($C^\intercal = B^\intercal A^\intercal$) para así extender su soporte también a `IP-N`, `OP-N` y `Gust-N` sin incluir hardware adicional.

Mientras que en otros aceleradores esta consideración solo tiene sentido a mayor nivel –en la fase de mapeo de SpMSpMs en el acelerador–, en Flexagon esta es una característica clave si se quiere obtener su máximo rendimiento, pues permite utilizar mapeos óptimos sin intervención de la CPU. Sin embargo, en el trabajo original no se llevó a cabo la evaluación de los *N-dataflows*.

Tabla II: Formatos de las matrices requeridas por cada *dataflow* soportado en Flexagon y concatenaciones entre capas de DNNs posibles. Los *M-dataflows* generan sus resultados en CSR mientras que los *N-dataflows* los generan en CSC. Dos *dataflows* son concatenables si el formato de salida (C) corresponde con el formato de entrada esperado (A). Se asume que los *weights* (B) se encuentran siempre disponibles en el formato correcto.

Dataflow	Formato			¿Concatenable?					
	A	*B*	*C*	IP-M	OP-M	Gust-M	IP-N	OP-N	Gust-N
IP-M	CSR	CSC	CSR	✓	✗	✓	✓	✗	✗
OP-M	CSC	CSR	CSR	✓	✗	✓	✓	✗	✗
Gust-M	CSR	CSR	CSR	✓	✗	✓	✓	✗	✗
IP-N	CSR	CSC	CSC	✗	✓	✗	✗	✓	✓
OP-N	CSC	CSR	CSC	✗	✓	✗	✗	✓	✓
Gust-N	CSC	CSC	CSC	✗	✓	✗	✗	✓	✓

Así, cada *dataflow* requiere que cada una de sus matrices A y B se encuentre codificada en CSR o CSC en función de si debe leerla por filas o por columnas. La Tabla II muestra cuales son los tipos de las matrices esperados en cada caso y el formato de la matriz C resultado. Nótese que si, por ejemplo, para `IP-M` tenemos las matrices A y B en formato CSR y CSC respectivamente, entonces intercambiar ambas matrices implica hacer el producto traspuesto (ya que A se interpretará como CSC y B como CSR, de acuerdo a lo explicado en la Subsec. II-B).

D.2 Concatenación de *Dataflows* en Flexagon

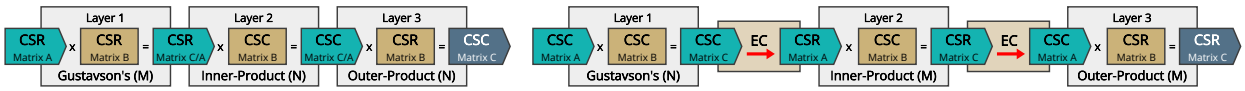
Soportar los seis *dataflows* en Flexagon permite la concatenación de resultados entre capas de DNN contiguas. Con concatenación nos referimos a cuando el formato de la matriz de salida (i.e., C) de una capa coincide con el formato de entrada (consideramos A para la explicación) requerido por el *dataflow* a utilizar para computar la siguiente capa. La Tabla II muestra cuales son los *dataflows* que son concatenables entre sí. La Fig. 4a muestra un ejemplo de cómo las capas pueden ser concatenadas seleccionando *dataflows* diferentes en cada una.

Si dos *dataflows* no pueden ser concatenados entre capas directamente, entonces es necesario realizar una Conversión Explícita o *Explicit Conversion* (EC) de formato de CSR a CSC o viceversa. La Fig. 4b muestra otro ejemplo donde se requiere de ECs para poder concatenar los distintos *dataflows* que se quieren ejecutar en cada capa. Por supuesto, una EC implica una sobrecarga en el rendimiento que requeriría de mecanismos específicos para reducir su impacto en el rendimiento global y/o evitar la intervención de la CPU durante la ejecución del modelo de DNN completo. Esta es una tarea complicada debido a la secuencialidad intrínseca al algoritmo de conversión y a que no se puede establecer un *pipeline* con el resto de fases de ejecución de Flexagon. No obstante, este aspecto tampoco fue abordado en el trabajo original.

III. Metodología Experimental

A. *Infraestructura de Simulación*

Para una evaluación detallada y precisa de la arquitectura de Flexagon utilizamos SST-STONNE [16], un simulador microarquitectural a nivel de ciclo de aceleradores para DNN. Este simulador se compone, por un lado, de STONNE,



(a) Usando *dataflows* concatenables (b) Usando *dataflows* no concatenables (se requieren ECs)

Fig. 4: Ejemplos de concatenación de *dataflows* en Flexagon.

Tabla III: Parámetros de configuración de Flexagon utilizados.

Parámetro	Valor
Frecuencia	1.00 GHz
Número de Multiplicadores (MN)	64
Número de Sumadores (MRN)	63
Ancho de banda de Distribución (DN)	16 elems/ciclo
Ancho de banda de Reducción/Unión (MRN)	16 elems/ciclo
Tamaño de Elemento (val + coord)	32 bits
Latencia de Acceso a L1 (*on-chip*)	1 ciclo
Tamaño de la FIFO L1 (*on-chip*)	256 bytes
Tamaño de la Caché L1 (*on-chip*)	1 MiB
Tamaño de la Línea de Caché L1 (*on-chip*)	128 bytes
Asociatividad de la Caché L1 (*on-chip*)	16
Número de Bancos de la Caché L1 (*on-chip*)	16
Tamaño de PSRAM	256 KiB
Tamaño de DRAM	16 GiB
Tiempo de acceso a DRAM / Ancho de banda	100 ns / 256 GB/s

encargado de la simulación de cada componente del acelerador; y SST [17], un popular *framework* para simulación arquitectural escalable de sistemas de *High-Performance Computing* del cual se obtienen los componentes que simulan una jerarquía de memoria que contiene la caché de Flexagon y una DRAM *off-chip* conectada por HBM 2.0. La Tabla III resume los principales parámetros de configuración del simulador usados para todas las evaluaciones. Esta configuración utilizada coincide con la misma utilizada en la publicación original de Flexagon con el fin de realizar una extensión y comparativa directa con los resultados originales. Además, tal y como se ilustra en la Fig. 5, Flexagon se considerará integrado dentro del chip de la CPU como un co-procesador compartiendo la caché de último nivel y la memoria principal con la CPU.

B. Evaluación de Modelos DNN Completos

Para la evaluación hemos considerado los mismos modelos DNN que en la evaluación original de Flexagon (ver Tabla IV). Todos estos modelos pertenecen a la suite de benchmarks MLPerf Inference [18], habiéndose seleccionado diferentes modelos DNN en cuanto al dominio de aplicación, tipo de capa y tamaño; concretamente los pertenecientes a la categoría para *edge devices*. En nuestra evaluación se estudian las seis variantes existentes de *dataflows*, tanto individualmente como agrupadas por el grupo de *dataflow* que implementan, y se comparan directamente con los resultados obtenidos en el trabajo original (que solo consideraba los *M-dataflows*).

C. Evaluación por Capas

Adicionalmente, también hemos evaluado un conjunto de seis capas seleccionadas a mano, mostradas en la Tabla V. Esta selección nos permitirá analizar con mayor detalle las principales características expuestas por cada *dataflow*, tratando de cubrir los principales casos de interés.

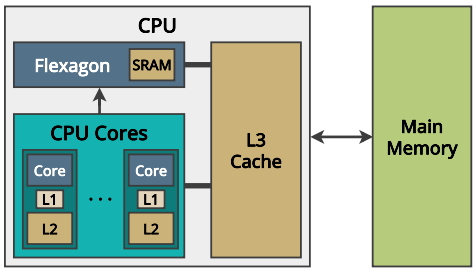


Fig. 5: Diagrama conceptual de la integración de Flexagon con la CPU como un co-procesador.

D. Estimación del Coste de las Explicit Conversions

En la publicación original no se consideraba el coste de las ECs. Como contribución de este trabajo, hemos estudiado distintas aproximaciones para estimar el coste de las EC en Flexagon. A continuación se detalla cada una de ellas:

Implementación Software en CPU. Como Flexagon se encuentra integrado en el mismo chip que la CPU, se puede delegar la tarea de realizar la EC a la CPU, la cual utilizará una implementación altamente optimizada del algoritmo de conversión para su estimación. Este diseño evita tener que introducir nuevo hardware adicional en el acelerador y se beneficia de compartir la jerarquía de memoria con la CPU, permitiendo así cambios de contexto extremadamente rápidos entre la CPU y el acelerador. Hemos seleccionado Intel MKL, concretamente su función `mkl_scsrcsc` (conversión de CSR a CSC y viceversa para precisión de 32-bits), y la suite `nanobench` [19] para perfilar el rendimiento de la EC a nivel de ciclo en este entorno. Las pruebas se han realizado sobre una CPU Intel Core™ i7-11700 a 2,50GHz, 8 núcleos físicos con HT, 384 KiB de caché L1, 4 MiB de caché L2 y 16 MiB de caché L3. Tras tomar medidas sobre un amplio abanico de ejemplos, hemos determinado que se requieren $\sim$ 34 ciclos de CPU en promedio por elemento no-cero de la matriz *sparse*. Si ajustamos este resultado a la frecuencia de Flexagon (1 GHz), obtenemos una estimación conceptual de $\sim$ 14 ciclos por elemento no-cero.

Implementación Hardware en MINT. Otra opción de diseño sería considerar un módulo específico hardware que permita realizar las ECs. Para esta evaluación hemos seleccionado MINT [6], el cual se encuentra integrado a nivel de controlador de memoria. Considerarlo introduce un área extra del $7,8\,\%$ (requiere de $0,41mm^2$ en comparación a los $5,28mm^2$ de nuestra configuración de Flexagon). En dicha propuesta se evalúa una mejora promedio en rendimiento de $\sim 4\times$ respecto a la implementación de Intel MKL. Por tanto, tomaremos como aproximación que realizar una EC empleando este módulo tomará $\sim$ 4 ciclos por elemento no-cero en Flexagon.

Tabla IV: Características de los modelos de DNN usados para la evaluación. Se incluyen los mismos modelos que los utilizados en la evaluación original de Flexagon [7]. Se consideran modelos de diferentes dominios: Clasificación de Imágenes (CI), Detección de Objetos (DO) y Procesamiento del Lenguaje Natural (NLP). *A* es la matriz de *weights* y *B* es la matriz de *inputs*.

Modelo de DNN	Dominio de Appl.	Nº de capas	Sparsity promedio		Tamaño promedio		Tamaño mínimo		Tamaño máximo	
			A	***B***	***A***	***B***	***A***	***B***	***A***	***B***
AlexNet (A)	CI	7	70 %	48 %	0.56 MiB	13.3 MiB	0.03 MiB	0.18 MiB	1.01 MiB	63.4 MiB
SqueezeNet (S)	CI	26	70 %	31 %	0.05 MiB	1.54 MiB	0.01 MiB	0.03 MiB	0.59 MiB	26.6 MiB
VGG-16 (V)	CI	8	90 %	80 %	0.44 MiB	5.70 MiB	0.001 MiB	0.44 MiB	0.90 MiB	17.7 MiB
ResNet-50 (R)	CI	54	89 %	52 %	0.20 MiB	1.31 MiB	0.002 MiB	0.008 MiB	0.99 MiB	26.6 MiB
SSD-ResNet (S-R)	DO	37	89 %	49 %	0.13 MiB	3.61 MiB	0.003 MiB	0.003 MiB	0.49 MiB	10.1 MiB
SSD-MobileNet (S-M)	DO	29	74 %	35 %	0.16 MiB	0.31 MiB	0.002 MiB	0.001 MiB	1.00 MiB	10.1 MiB
DistilBERT (DB)	NLP	36	50 %	0.04 %	2.25 MiB	0.35 MiB	1.13 MiB	0.23 MiB	4.50 MiB	0.94 MiB
MobileBERT (MB)	NLP	316	50 %	11 %	0.11 MiB	0.01 MiB	0.03 MiB	0.04 MiB	0.38 MiB	0.02 MiB

Tabla V: Características de las capas de DNN utilizadas en la evaluación. La columna "*Dataflow objetivo*" indica cual es el *dataflow* más relevante para analizar en cada capa. *A* es la matriz de *weights* y *B* es la matriz de *inputs*.

Capa de DNN	***Dataflow*** objetivo	M	N	K	Sparsity		Tamaño	
					A	***B***	***A***	***B***
SqueezeNet_2 (S2)	IP-M	64	16	2916	68 %	12 %	0.001 MiB	0.16 MiB
SSD-MobileNet_0 (S-M0)	IP-N	64	32	22500	73 %	41 %	0.002 MiB	1.6 MiB
VGG-16_0 (V0)	OP-M	64	27	49284	85 %	0 %	0.001 MiB	5.1 MiB
ResNet-50_51 (R51)	OP-N	512	4608	25	89 %	79 %	1.0 MiB	0.09 MiB
DistilBERT_1 (DB1)	Gust-M	768	768	80	50 %	0 %	1.1 MiB	0.23 MiB
AlexNet_4 (A4)	Gust-N	256	2304	121	70 %	83 %	0.68 MiB	0.18 MiB

IV. Caracterización de Dataflows en Flexagon

A. Análisis de dataflows para SpMSpM

Como se ha mencionado anteriormente, el *dataflow* óptimo para la ejecución de una capa de DNN puede variar drásticamente de una capa a otra incluso dentro del mismo modelo. En el trabajo original únicamente se evaluaban los *M-dataflows*. En la Fig. 6 los comparamos directamente con los nuevos resultados que incluyen los *N-dataflows*, en la cual cuantificamos el factor de mejora que implica considerar todas las variantes de *dataflows* como uno de los elementos clave de Flexagon. En ella mostramos el rendimiento, en ciclos (menor es mejor), normalizados respecto a los resultados originales sin considerar el impacto de las ECs requeridas.

Como se puede observar, considerar también los *N-dataflows* habilita una mejora de hasta $1,92\times$ en el mejor caso (AlexNet), obteniendo una mejora promedio de 1,51× frente a los resultados originales de Flexagon (`Flexagon-M`). También podemos observar otros ejemplos interesantes en los resultados, como que `Gust-N` es $1,48\times$ más eficiente en promedio que `Gust-M` para todos los casos excepto DistilBERT, donde es $8,83\times$ veces más lento. Para IP obtenemos unos resultados similares, siendo `IP-N` $1,21\times$ más rápido en promedio que `IP-M`. Sin embargo, con OP sucede que `OP-M` es $1,26\times$ más eficiente en promedio que su versión opuesta `OP-N`. Con esto demostramos la importancia de considerar las seis variaciones de *dataflows* a la hora de diseñar una estrategia de mapeo óptima y específica para Flexagon, pues las diferencias de rendimiento entre los *M-dataflows* y los *N-dataflows* en cada caso son significativas.

Adicionalmente, la Fig. 7 muestra el rendimiento de Flexagon comparado a otras propuestas de aceleradores que soportan únicamente un tipo de *dataflow* (e.g. SIGMA [5], SpArch [9] y GAMMA [10] para IP, OP y Gust respectivamente). Dado que cada arquitectura tiene una implementación distinta nos referiremos a ellas únicamente por el *dataflow* que implementan, siendo todos los resultados relativos a la implementación de Flexagon. Nótese que, en los resultados, cada arquitectura siempre selecciona el *dataflow* óptimo entre sus opciones disponibles (*M-dataflows* y *N-dataflows*).

Los resultados muestran que utilizar todos los *dataflows* (i.e., `Flexagon`) obtiene una mejora de rendimiento promedio de $4,21\times$, $2,58\times$ y $1,01\times$ frente a `IP-only`, `OP-only` y `Gust-only` respectivamente. Estos resultados muestran que Flexagon se beneficia principalmente de los *dataflows* basados en Gust y que no considerar los demás *dataflows* únicamente implicaría una pérdida de rendimiento del $\sim 1,4\,\%$ respecto al rendimiento ideal para los modelos de DNN evaluados. Esto posibilita nuevas oportunidades de exploración en el desarrollo de Flexagon, como eliminar la lógica adicional de los controladores de memoria para IP y OP o ahorrar área y consumo energético reduciendo el tamaño de la PSRAM (que requiere de un tamaño mayor para el soporte a OP).

Por último, la Fig. 8 muestra el porcentaje total de los *dataflows* óptimos seleccionados en las capas de cada modelo, donde se hace evidente que los *dataflows* basados en Gust predominan en el mapeo óptimo de Flexagon. Aunque en algunos de ellos se incluyen *dataflows* IP o OP dentro de los óptimos, ya hemos mostrado que su impacto final en el rendimiento es mucho menor que el que produce Gust en el resto de capas.

B. Evaluación por Capas

Para comprender mejor las razones que subyacen a los beneficios obtenidos por cada *dataflow* para cada modelo DNN, es necesario realizar un análisis de grano fino más exhaustivo y detallado. Aquí analizamos un conjunto de seis capas representativas pertenecientes a los modelos DNN anteriores. Las principales características de cada capa seleccionada se muestran en Tabla V.

Para la evaluación, nos basamos en múltiples gráficas. La Fig. 9 muestra una comparación del rendimiento de cada *dataflow* para cada capa (los resultados se normalizan al mejor caso en cada capa). La Fig. 10 muestra la cantidad de tráfico *on-chip* que fluye a través de la jerarquía de memoria de Flexagon para cada *dataflow*. Por último, la Fig. 11 muestra la tasa de fallos de la caché para los accesos a la matriz *streaming*. Nótese que en ellos solo se incluyen los resultados obtenidos por cada *dataflow*, por lo que Flexagon obtiene implícitamente el resultado del que obtiene mejor rendimiento.

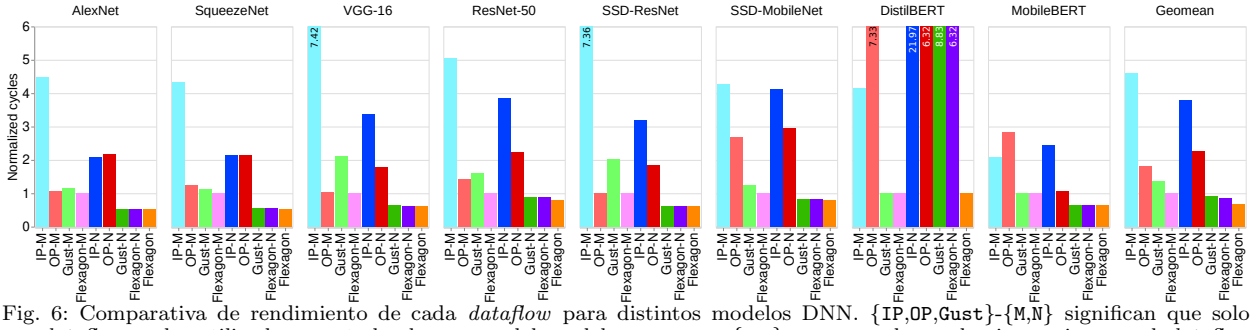


Fig. 6: Comparativa de rendimiento de cada *dataflow* para distintos modelos DNN. {`IP`,`OP`,`Gust`}-{`M`,`N`} significan que solo ese *dataflow* se ha utilizado para todas las capas del modelo. `Flexagon`-{`M`,`N`} corresponde a seleccionar siempre el *dataflow* óptimo dentro del grupo de los {M,N}-*dataflows* respectivamente. `Flexagon` selecciona siempre el *dataflow* óptimo entre las seis opciones posibles. Los resultados, en ciclos, se encuentran normalizados respecto a los del trabajo original (`Flexagon-M`). No se considera el coste de las conversiones de formato requeridas entre capas.

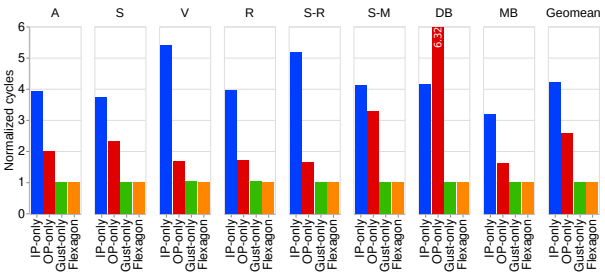


Fig. 7: Comparativa de rendimiento de cada arquitectura para distintos modelos DNN. Los resultados, en ciclos, están normalizados a `Flexagon`. `IP-only` cubre las arquitecturas que únicamente soportan *dataflows* IP; igual para `OP-only` y `Gust-only`. `Flexagon` siempre selecciona el mejor *dataflow* entre los seis disponibles. No se considera el coste de las conversiones de formato requeridas entre capas.

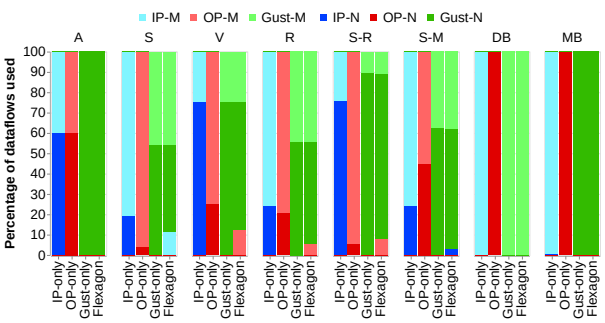


Fig. 8: Porcentaje de *dataflows* óptimos utilizados en cada arquitectura para distintos modelos DNN. Los porcentajes se calculan en base al número de capas del modelo que utilizan cada *dataflow* respecto al total de capas del modelo.

Resultados de *Inner-Product*. Comenzamos el análisis hablando de la capa S2, que se beneficia principalmente del uso de `IP-M`. En este caso, podemos ver que tanto `IP-M` como `IP-N` obtienen casi el doble de rendimiento que los otros *dataflows*, lo cual es gracias a que, al ser tan pequeñas las matrices implicadas, no se produce apenas tráfico de *psums* y se pueden enviar sumas completas directamente a memoria (i.e., si para una operación SpMSpM la dimensión K es menor que el número de multiplicadores en el MN, entonces se garantiza que siempre se puede producir una suma completa). Esto último también se aplica a las capas S-M0 y V0, donde los flujos de datos IP obtienen un buen rendimiento en general, y en las capas R51, DB1 y A4, donde se obtiene un rendimiento muy pobre por tener un valor de su dimensión K muy grande.

Generalmente, en los *dataflows* de tipo IP el mapeo que obtiene el rendimiento óptimo es aquel que establece como *stationary* la matriz de mayor tamaño, pues seleccionarla como matriz *streaming* implicaría un mayor número de accesos a memoria y una mayor tasa de fallos de caché. Cabe remarcar que, asumiendo M-*dataflows*, la matriz B deberá ser leída un total de M veces (i.e., número de filas de la matriz A). Por tanto, en teoría, deberíamos seleccionar `IP-M` si se satisface que $M \times Tamaño\_B \leq N \times Tamaño\_A$ o `IP-N` en caso contrario. Se puede observar que esto se cumple para la capa S-M0 (que se beneficia de `IP-N`), donde la tasa de fallos de `IP-M` es muy superior a la de `IP-N` (3,1 % y 0,3 % respectivamente) ya que la matriz B no cabe completamente en la caché para la matriz *streaming* en el caso de `IP-M`. Esto también sucede en las capas V0, R51, DB1 y A4.

En general, los *dataflows* basados en IP pueden funcionar mejor siempre que las matrices sean suficientemente pequeñas y densas ($\lesssim$ 10 KiB con $\sim$ 10 % de *sparsity*). Sin embargo, si una de las matrices tiene un tamaño mayor entonces aumenta el número de intersecciones que hay que comprobar, disminuyendo enormemente el rendimiento (e.g., esto ocurre en las capas S-M0, V0 y R51). La mayor limitación de este *dataflow* es que, dado que solo es aplicable de forma óptima a matrices pequeñas, su margen de optimización en un modelo DNN final es muy pequeño (mejoras del orden de 10^5 ciclos, mientras que en el resto de *dataflows* se observan mejoras del orden de hasta 10^7 ciclos), lo cual se refleja en la cantidad de tráfico de memoria que producen las capas que se benefician de IP comparadas al resto.

Resultados de *Outer-Product*. Cuando operamos con matrices más grandes, podemos ver que tanto los *dataflows* OP como los Gust proporcionan resultados mucho mejores que los basados en IP (e.g., V0, R51, DB1, A4). Se puede observar que tanto `OP-M` como `OP-N` producen casi la misma cantidad de tráfico *on-chip* y que tiene una tasa de fallos de caché muy similar para todas las capas. Además, también podemos ver que `OP-M` siempre obtiene mejor rendimiento que `OP-N` cuando la matriz *stationary* (B) es mayor que la matriz *streaming* (A) y viceversa (al contrario que IP). La razón de establecer la matriz más grande como la *streaming* se debe a las diferen-

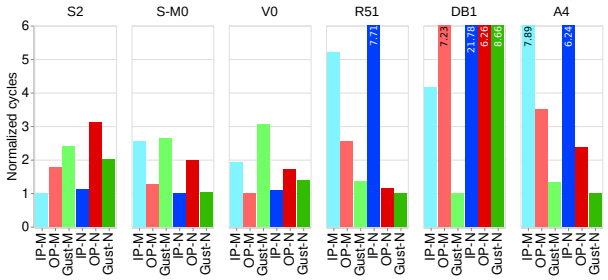


Fig. 9: Comparativa de rendimiento de cada *dataflow* para cada una de las capas de DNN seleccionadas. Los resultados se encuentran normalizados al mejor caso en cada capa.

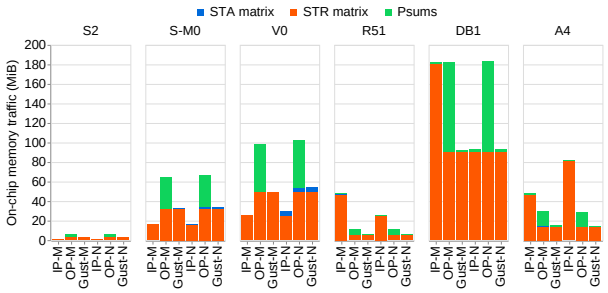


Fig. 10: Tráfico de memoria (en MiB) que fluye a través de la memoria *on-chip* de Flexagon en cada *dataflow* para cada una de las capas de DNN seleccionadas. `STA` y `STR` se refieren a las matrices *stationary* y *streaming* respectivamente.

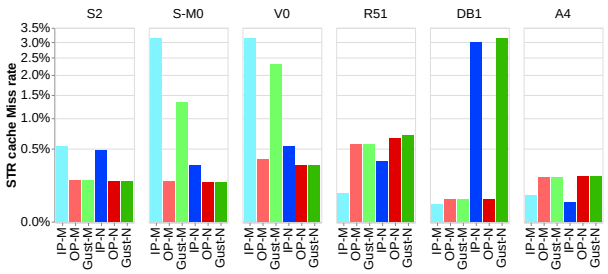


Fig. 11: Tasa de fallos de caché para la matriz *streaming* en cada *dataflow* en cada una de las capas de DNN seleccionadas.

cias internas que presenta cada estructura SRAM. Mientras que la caché puede producir fallos de acceso y, por tanto, latencias de acceso mayores, la FIFO dispone de los datos encolados con anterioridad, pudiendo accederse con latencia constante.

Resultados de *Gustavson's*. Al igual que ocurre con los *dataflows* IP, los basados en Gust se benefician de establecer su matriz mayor como *stationary*, como puede observarse en los resultados de las capas DB1 (gana `Gust-M`) y A4 (gana `Gust-N`). En concreto, en el caso `Gust-N` de la capa DB1, la matriz *streaming* no cabe por completo en la caché, por lo que la tasa de fallos se dispara y se produce una enorme penalización en el rendimiento (8,66× veces más ineficiente que `Gust-M`).

Si comparamos Gust con los otros *dataflows* podemos observar que, aunque los *dataflows* basados en IP son ideales para matrices pequeñas ($\lesssim$ 10 KiB), los basados en Gust funcionan realmente muy bien también para estos casos (véanse los ejemplos S2 y S-M0) y, en los casos en que es peor, la sobrecarga introducida no es demasiado notable ($\sim 10^5$ ciclos) en comparación con el número total de ciclos necesarios para calcular todo el modelo DNN ($\sim 10^8$ ciclos). Cuando se comparan con los *dataflows* basados en OP, podemos encontrar una propiedad interesante: los basados en Gust rinden mejor ($1,96\times$ en promedio para las capas seleccionadas) cuando tienen la misma tasa de fallos de caché que los basados en OP (i.e., la matriz *streaming* se puede almacenar completamente en caché, por lo que solo es necesario acceder a cada dato en memoria principal una vez como OP). Esto se debe a que, a pesar de que comparten latencias de acceso a memoria prácticamente idénticas en estas situaciones, en OP el coste de realizar la reducción de las matrices parciales que se producen es mucho más costoso que el de reducir fibras parciales como hacen los *dataflows* Gust, lo que implica una diferencia significativa de tráfico de memoria *on-chip* (e.g., OP requiere $1,97\times$ y $1,86\times$ más tráfico *on-chip* que Gust para DB1 y A4 respectivamente). Una excepción a este caso es la capa S0, donde las fibras parciales son tan pequeñas que, a pesar de tener OP y Gust la misma tasa de fallos de caché, no se consigue hacer un uso completo simultáneo de todas las unidades de la MRN, siendo OP un poco más eficiente en este caso (en el orden de 10^5 ciclos)

Como se puede ver, los *dataflows* basados en Gust conservan algunas de las mejores ventajas tanto de IP y OP y las hace una opción mucho más polivalente para la mayoría de capas. Por un lado, las diferencias de rendimiento respecto a IP, en los casos donde este gana, son de órdenes de magnitud muy pequeños ($\sim 10^5$ ciclos) en comparación al total de ciclos que requiere un modelo DNN completo para ejecutarse ($\sim 10^8$ ciclos). En el resto de casos, Gust es en promedio $4,21\times$ más eficiente como se ha visto anteriormente en la Fig. 7. Esto hace que resulte asumible utilizar siempre un *dataflow* Gust antes que IP para cualquier capa (alternando entre su versión *M-dataflow* o *N-dataflow* según corresponda). Por otro lado, también hemos visto que, en caso de poder almacenarse completamente alguna de las dos matrices en la caché *on-chip*, Gust es también en casi todos los casos una mejor opción que OP, obteniendo una mejora promedio de $2,58\times$. Finalmente, si esto último no se cumple entonces los *dataflows* OP serían más apropiados pues maximizan el reuso de datos y minimizan el tráfico *off-chip* (el cual supone la mayor limitación del resto de arquitecturas). No obstante, dependiendo del modelo DNN se emplean matrices de distintos tamaños, aunque la tendencia general es a utilizar matrices que no superen los $\sim$ 10 MiB, siendo lo ideal ajustar la memoria del acelerador para cubrir estos requisitos. Por tanto, esto queda abierto para trabajo futuro como una cuestión de implementación del acelerador en función de los requerimientos finales de la aplicaciones a las que se destine.

V. Caracterización del Impacto de las Explicit Conversions

El trabajo original no estudiaba el impacto de las ECs en el rendimiento global de la ejecución de un modelo DNN ya que únicamente se consideraban los *M-dataflows*, siendo estos fácilmente concatenables. En esta sección evaluamos su impacto real y como,

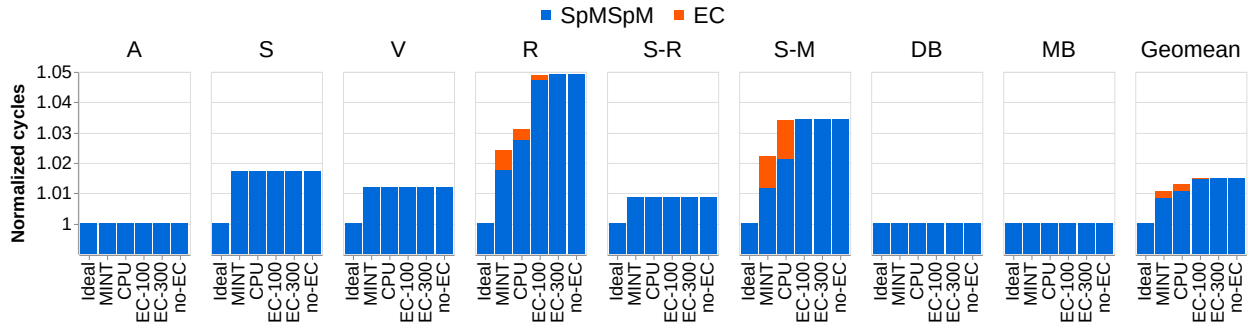


Fig. 12: Comparativa de rendimiento de diferentes mapeos óptimos considerando diferentes costes para las ECs para distintos modelos DNN. Los resultados, en ciclos, están normalizados respecto a `Ideal`. Las estimaciones son `Ideal` con $EC_{coste} = 0$ *ciclos*, `MINT` con $EC_{coste} = 4$ *ciclos*, `CPU` con $EC_{coste} = 14$ *ciclos*, `EC-100` con $EC_{coste} = 100$ *ciclos*, `EC-300` con $EC_{coste} = 300$ *ciclos* y `no-EC` con $EC_{coste} = \infty$ *ciclos*.

cuando se consideran, pueden afectar a los mapeos óptimos de los modelos.

A. Búsqueda de Mapeos Óptimos para Modelos DNN

A la hora de ejecutar un modelo DNN completo sobre Flexagon es importante determinar cual va a ser el *dataflow* óptimo para cada una de sus capas como ya se ha visto en la Sec. IV. Sin embargo, hasta el momento siempre hemos considerado que las conversiones de formato que se requieren para poder concatenar los *dataflows* tienen $EC_{coste} = 0$ *ciclos*, seleccionando siempre aquel *dataflow* que ofrece mejor rendimiento. Al considerar ahora dichos costes, los mapeos pueden verse influenciados. En general, siempre se preferirá utilizar *dataflows* concatenables entre sí (véase la Tabla II) para evitar tener que realizar ECs, pero habrá situaciones donde será beneficioso realizarlas para así poder ejecutar un *dataflow* específico y óptimo durante la siguiente capa.

Para esta evaluación consideraremos los mapeos ideales y óptimos que, teniendo en cuenta los costes de las ECs, podría realizar un mapeador si dispusiera previamente de los tiempos de ejecución de cada capa. Teniendo el número de ciclos que requiere cada *dataflow* para cada capa de un modelo DNN, con un algoritmo de programación dinámica obtenemos el *Camino de Coste Mínimo*. Este algoritmo opera en $\Theta(n \times m)$, donde n es el número de capas dentro del modelo y m el número de posibles *dataflows* (en este caso, 6). Un mapeador que sea capaz de realizar esta tarea mediante análisis estático y/o dinámico queda fuera del objetivo de este trabajo, así que únicamente tomaremos este modelo ideal como referencia.

B. Análisis de Rendimiento Considerando ECs

Como ya se ha expuesto en la Subsec. III-D, consideramos diferentes estimaciones del coste de realizar una EC en función del número de elementos no-cero de cada matriz para distintas implementaciones de Flexagon. Así, consideramos cuatro versiones diferentes: `Ideal` ($EC_{coste} = 0$ *ciclos*; utilizado en las evaluaciones anteriores de la Sec. IV), `MINT` ($EC_{coste} = 4$ *ciclos*; Flexagon junto al coacelerador), `CPU` ($EC_{coste} = 14$ *ciclos*; Flexagon integrado junto a la CPU) y `no-EC` ($EC_{coste} = \infty$ *ciclos*; se evitan las EC). Además, incluimos dos versiones adicionales, `EC-100` y `EC-300`, donde consideramos $EC_{coste} = 100$ *ciclos* y $EC_{coste} = 300$ *ciclos* respectivamente con el fin de ofrecer un análisis de sensibilidad más completo. Como se observará más adelante, nuestras evaluaciones muestran que el mapeo óptimo en los escenarios donde se considera $EC_{coste} = 300$ *ciclos* son prácticamente idénticos a los que se obtienen cuando se evitan completamente ya que introducen una sobrecarga adicional que no es asequible.

La Fig. 12 muestra los resultados obtenidos para cada versión utilizando para cada modelo el mapeo óptimo. En la figura se muestra, normalizado respecto a `Ideal`, el número de ciclos frente al total que se dedican a computar SpMSpMs y cuánto se invierte en realizar las ECs. Rápidamente podemos observar que considerar las EC no introduce una sobrecarga significativa sobre el rendimiento global, introduciendo únicamente un $1,5\,\%$ de ciclos extra en promedio en el peor de los casos (i.e., evitarlas completamente). Esta es una de las principales ventajas de considerar las seis variantes de *dataflows* como uno de los pilares fundamentales del uso de Flexagon: la adaptabilidad a las distintas cargas de trabajo y la flexibilidad en el concatenamiento de los formatos. En los mapeos óptimos se explota a concatenabilidad de `Gust-M` y `Gust-N` y se aprovechan las conversiones de formato implícitas que ofrecen `OP-M` y `OP-N`, permitiendo así obtener un rendimiento cercano al ideal.

Más en detalle, podemos ver que hay casos en los que el mapeo ideal coincide con el mapeo que evita las ECs (Alexnet, DistilBERT y MobileBERT). También hay otros modelos en los que aplicar una EC es tan costoso que, aun considerando MINT, implica automáticamente utilizar como mapeo óptimo uno que evita completamente las ECs (SqueezeNet, VGG-16 y SSD-ResNet). Sin embargo, son de especial interés los casos de ResNet-50 y SSD-MobileNet, donde sí que varía el rendimiento en función de las distintas implementaciones de Flexagon y el coste de las EC. En ellas se aplican ECs en el $9,4\,\%$ y en el $10,7\,\%$ de transiciones entre capas respectivamente, significando tan solo desde un $0,3\,\%$ hasta un $1,2\,\%$ de ciclos en total en comparación a las SpMSpMs realizadas en la ejecución del modelo. Hay dos aspectos que reseñar: i) aun en el caso de realizarlas, su im-

pacto en el rendimiento es realmente muy pequeño ($\sim 1\,\%$) y ii) a partir de $EC_{coste} = 100$ *ciclos* el rendimiento es prácticamente idéntico que si se evitan realizar completamente las EC.

Así, demostramos que el impacto de las EC sobre Flexagon no es un factor determinante pues la adaptabilidad que ofrece para las distintas cargas de trabajo y su flexibilidad en el concatenamiento de *dataflows* permite, en la gran mayoría de situaciones, poder evitarlas y que el rendimiento del acelerador se acerque al ideal (solo se requiere un $1,5\,\%$ de ciclos adicionales en promedio).

VI. Conclusiones

En este trabajo hemos extendido la evaluación original del acelerador hardware Flexagon. Por un lado, hemos evaluado la conveniencia de considerar las seis principales variantes de *dataflows* para operaciones SpMSpM, habiendo obtenido una mejora promedio de rendimiento de $1,51\times$ frente a considerar únicamente los *M-dataflows*. Por otro lado, hemos comparado detalladamente las principales diferencias y casos de uso de cada *dataflow*, habiendo determinado que Gust es la opción más adaptable a la mayoría de capas de modelos DNN y que utilizarlo exclusivamente solo implicaría una pérdida de rendimiento del $1,4\,\%$ respecto a considerar los tres principales grupos de *dataflows*. Por último, hemos propuesto y evaluado distintos mecanismos hardware para dar soporte a las ECs de manera eficiente en Flexagon, habiendo concluido que el sobrecoste introducido por estas operaciones varía entre el $1{,}0\,\%$ y el $1{,}5\,\%$ en promedio respecto al rendimiento global para la ejecución de un modelo DNN.

Agradecimientos

Trabajo realizado en el contexto del proyecto PID2022-136315OB-I00 financiado por MCIN/AEI/10.13039/501100011033/ y "FEDER Una manera de hacer Europa", UE. También en el contexto de la ayuda RYC2021-031966-I financiada por MCIN/AEI/ 10.13039/501100011033 y por la "Unión Europea NextGenerationEU/PRTR". Adrián Navarro ha sido financiado mediante la ayuda 22294/FPI/23 de Fundación Séneca, Agencia de Ciencia y Tecnología de la Región de Murcia.

Referencias

[1] Salvatore Filippone et al., "PSBLAS: A Library for Parallel Linear Algebra Computation on Sparse Matrices," *ACM Trans. Math. Softw.*, vol. 26, no. 4, pp. 527–550, 2000.

[2] Narayanan Sundaram et al., "GraphMat: High Performance Graph Analytics Made Productive," *Proc. VLDB Endow.*, vol. 8, no. 11, pp. 1214–1225, 2015.

[3] Valéry Weber et al., "Semiempirical Molecular Dynamics (SEMD) I: Midpoint-Based Parallel Sparse Matrix–Matrix Multiplication Algorithm for Matrices with Decay," *Journal of Chemical Theory and Computation*, vol. 11, no. 7, pp. 3145–3152, 2015.

[4] Tom B. Brown et al., "Language Models Are Few-Shot Learners," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*. 2020, NIPS'20, Curran Associates Inc.

[5] Eric Qin et al., "SIGMA: A Sparse and Irregular GEMM Accelerator with Flexible Interconnects for DNN Training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 58–70.

[6] Eric Qin et al., "Extending Sparse Tensor Accelerators to Support Multiple Compression Formats," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2021, pp. 1014–1024.

[7] Francisco Muñoz Martínez et al., "Flexagon: A Multi-Dataflow Sparse-Sparse Matrix Multiplication Accelerator for Efficient DNN Processing," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 2023, ASPLOS 2023, p. 252–265, Association for Computing Machinery.

[8] Subhankar Pal et al., "OuterSPACE: An Outer Product Based Sparse Matrix Multiplication Accelerator," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 724–736.

[9] Zhekai Zhang et al., "SpArch: Efficient Architecture for Sparse Matrix Multiplication," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 261–274.

[10] Guowei Zhang et al., "Gamma: Leveraging Gustavson's Algorithm to Accelerate Sparse Matrix Multiplication," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 2021, ASPLOS '21, p. 687–701, Association for Computing Machinery.

[11] Zhiyao Li et al., "Spada: Accelerating Sparse Matrix Multiplication with Adaptive Dataflow," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 2023, ASPLOS 2023, p. 747–761, Association for Computing Machinery.

[12] Maxime Oquab et al., "DINOv2: Learning Robust Visual Features without Supervision," 2023.

[13] Robin Rombach et al., "High-Resolution Image Synthesis with Latent Diffusion Models," 2022.

[14] Vivienne Sze et al., "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[15] Nitish Srivastava et al., "MatRaptor: A Sparse-Sparse Matrix Multiplication Accelerator Based on Row-Wise Product," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 766–780.

[16] Francisco Muñoz-Martínez et al., "SST-STONNE: Enabling Cycle-Level Simulation of Flexible Spatial Accelerators for DNNs and GNNs with a Detailed Memory Hierarchy," in *Workshop on Modeling & Simulation of Systems and Applications (ModSim)*, 2022.

[17] A. F. Rodrigues et al., "The Structural Simulation Toolkit," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 4, pp. 37–42, 2011.

[18] Peter Mattson et al., "MLPerf: An Industry Standard Benchmark Suite for Machine Learning Performance," *IEEE Micro*, vol. 40, no. 2, pp. 8–16, 2020.

[19] "nanobench: Simple, fast, accurate single-header microbenchmarking functionality for C++11/14/17/20," `https://github.com/martinus/nanobench`.