Evaluación de un Sistema de Recomendación en un Acelerador Híbrido

Nicolás Meseguer-Iborra,[1] Francisco Muñoz-Martínez[2], Manuel E. Acacio[2], José L. Abellán[1]

Resumen— **Los sistemas de recomendación basados en técnicas de *Deep Learning* (DL) es un área en constante evolución. Los sistemas actuales han de ser capaces de generar una salida (por ejemplo, la probabilidad de hacer clic sobre un determinado anuncio) de la manera más precisa (volver a mostrarlo o no) y dentro de un estricto margen de tiempo para que las empresas digitales maximicen sus beneficios. Sin embargo, cuando estos modelos ya entrenados se despliegan sobre Datacenters con un gran volumen de datos a procesar (cientos de miles de usuarios haciendo clic sobre anuncios), el tiempo de cómputo y la demanda de recursos computacionales (unidades de cómputo, consumo de memoria, etc.) aumenta considerablemente. Como resultado de esto, surge la necesidad de crear nuevas arquitecturas aceleradoras especializadas en procesar sistemas de recomendación. En este trabajo se va a realizar una caracterización de la ejecución del sistema de recomendación DLRM (Facebook) sobre una arquitectura aceleradora unificada híbrida (arquitectura para cómputo *sparse* y otra para cómputo *dense*; etapas en la ejecución de DLRM) que hemos propuesto para acelerar esta carga de trabajo. Para ello se usará STONNE, un simulador arquitectural para aceleradores de DL. La evaluación de DLRM se realizará mediante cargas de trabajo similares a las esperadas en un entorno real. Este análisis permitirá detectar cuellos de botella clave en la ejecución de DLRM, de modo que nos permita, como trabajo futuro, el diseño de un acelerador más optimizado para DLRM.**

Palabras clave— **Sistema de recomendación, DLRM, Caracterización de rendimiento, Acelerador para Deep Learning, Herramienta de simulación.**

I. Introducción

LOS sistemas de recomendación han emergido como herramientas clave para abordar tareas de personalización y recomendación [1]; como por ejemplo, vídeos que podrían interesarte, personas que quizá conozcas o anuncios basados en tus intereses. De esta forma, a día de hoy, existe la necesidad de recomendar una gran cantidad de datos (cientos de *gigabytes*) a un público determinado de usuarios dentro de un estricto margen de tiempo. Es por ello que se emplean grandes y costosos Datacenters para su ejecución con multitud de nodos heterogéneos (compuestos por CPU+FPGA/GPU) [2].

Para alcanzar la mayor eficiencia computacional posible en la ejecución de estos sistemas de recomendación, tanto desde el punto de vista del rendimiento (tiempo en realizar una recomendación) así como en cuanto a consumo energético, es necesario realizar un co-diseño SW/HW. De este modo, una vez conocidas las características de ejecución (e.g., flujo de datos, tipos de operaciones, cuellos de botella existentes, etc) del sistema de recomendación (SW), se está en disposición de comprender cómo mejorar el diseño/arquitectura de una plataforma de cómputo especializada (HW) que acelere su procesamiento. Es decir, desarrollar un acelerador de dominio específico (del inglés *Domain-Specific Accelerator* o DSA) para sistemas de recomendación.

Desde una perspectiva algorítmica (SW), hasta hace poco los sistemas de recomendación hacían uso de técnicas como *content filtering* [3] y *collaborative filtering* [4] (ver Sección II), las cuales se basan en métodos como *k-vecinos* o *asociación por grupos*, logrando obtener resultados aceptables [5], [6]. Sin embargo, recientemente los sistemas de recomendación han empezado a hacer uso de técnicas de *aprendizaje profundo* (del inglés *deep-learning* o DL), integrando en sus modelos redes neuronales profundas (del inglés, *Deep Neural Networks* o DNNs), que superan ampliamente la precisión obtenida en los modelos de recomendación tradicionales mencionados anteriormente [1].

Las DNNs tienen dos fases de ejecución: una primera fase de entrenamiento o *training*, donde se entrena la red para que aprenda a realizar una determinada tarea (e.g., detección de señales viales en una carretera), ajustando los pesos de la red neuronal; y una segunda fase de predicción o inferencia, donde el modelo DNN se despliega para ser utilizado (por ejemplo, que la DNN entrenada se instale en una cámara en un coche y ayude en la conducción automática del vehículo). En este trabajo nos vamos a centrar en estudiar la fase de inferencia de los sistemas de recomendación.

Con el objetivo de maximizar el rendimiento-por-vatio en la ejecución de la fase de inferencia, actualmente están desarrollándose multitud de co-diseños SW/HW, propiciando el desarrollo de una gran cantidad de DSAs para DL [7], [8], [9], [10], [11], [12]. Un ejemplo claro de estas propuestas es la *Tensor Processing Unit* o TPU [7] de Google, que maximiza el rendimiento-por-vatio en el procesamiento de las operaciones de multiplicación de matrices (clave en procesamiento de DNNs) mediante una arquitectura sistólica.

Los modelos de recomendación actuales como DLRM difieren de otros modelos de DL por la necesidad de tratar con ingentes cantidades de variables categóricas (e.g., la categoría de los vídeos que el usuario ha visualizado, el género de las películas, tipo de anuncios, etc), entendidas como *sparse features* porque normalmente se representan en formato vector/matriz y constan de una gran cantidad de ceros

[1]Dpto. de Grado en Ingeniería Informática, Universidad Católica de Murcia, e-mail: {nmeseguer2,jlabellan}@ucam.edu.
[2]Dpto. de Ingeniería y Tecnología de Computadores, Universidad de Murcia, e-mail: {francisco.munoz2,meacacio}@um.es.

(es decir, una matriz dispersa o *sparse*). Por otro lado, el modelo DLRM también utiliza variables continuas (e.g., la matriz en DLRM que contiene los pesos ya entrenados, historial de estadísticas, predicciones de otros modelos [13], etc.) que constituyen *dense features* (matriz o vector con pocos ceros).

Debido a los grandes retos que se presentan a la hora de ejecutar este modelo DLRM (e.g., patrón de accesos a memoria irregular a la hora de acceder a las *sparse features*, heterogeneidad de las operaciones sparse y dense, etc) y a la tremenda popularidad y demanda que está teniendo este modelo, recientemente se han presentado algunos trabajos de caracterización que tratan de averiguar cómo se comporta este modelo DLRM en arquitecturas tradicionales como CPUs o GPUs [2], [14]. Sin embargo, todavía no se ha realizado una caracterización de ejecución detallada de su fase de inferencia sobre dispositivos DSA aceleradores empotrados. Éste constituye el objetivo principal de este trabajo.

En particular, dado el procesamiento *sparse* y *dense* que requiere DLRM, vamos a basar la caracterización en una arquitectura aceleradora híbrida especializada en procesamiento *sparse* y *dense*. En particular, el diseño utilizado está inspirado en HyGCN [15], un arquitectura acelerador para procesamiento de GCNs (*Graph Convolutional Neural Networks*) que integra dos aceleradores dedicados para distinto tipo de cómputo (*sparse* y *dense*) dentro de un mismo chip con el fin de obtener alto rendimiento y bajo consumo de energía. En nuestro caso, la arquitectura evaluada combina en un mismo acelerador la arquitectura MAERI [8], para el cómputo *dense*, y la arquitectura SIGMA [9], para el cómputo *sparse*.

Para poder realizar la caracterización de DLRM sobre este acelerador para inferencia propuesto, vamos a utilizar el simulador STONNE [16], que nos permite simular a nivel de ciclo distintos tipos de arquitecturas aceleradoras para DL (actualmente, MAERI, SIGMA y TPU), y permite realizar simulaciones del procesamiento de DNNs reales y completas ya que STONNE está conectado con el framework para DL *PyTorch*.

Para nuestro estudio, hemos realizado un análisis detallado de la ejecución de DLRM sobre el acelerador híbrido propuesto mediante cargas de trabajo sintéticas similares a las que se producen en entorno real, utilizando métricas como el tiempo de ejecución y el número de accesos a memoria. En concreto, en este trabajo hemos descubierto el fuerte peso que representa el cómputo de las variables categóricas, *sparse*, ya que el cómputo del resto del modelo varía en función del tamaño de éstas.

II. Background y Trabajo Relacionado

A. El sistema de recomendación DLRM

Los sistemas de recomendación se usan en la actualidad para una gran variedad de tareas en grandes compañías, incluyendo tasas de clic por anuncio (CTR), predicciones, *rankings*, etc. Tradicionalmente, dos son las perspectivas que se han utilizado para el desarrollo de modelos de personalización y recomendación. La primera viene de las diferentes técnicas de los sistemas de recomendación, como *content-based filtering*, en los que si un usuario A ve dos vídeos de animales, el sistema le recomienda vídeos de animales; y *collaborative filtering*, si un usuario A es similar a un usuario B (por los vídeos que ha visto) y el usuario A indica que le gusta un vídeo, entonces el vídeo se le mostrará al usuario B. Pero en los últimos años se ha comenzado a hacer uso de técnicas híbridas que combinan *content-based filtering* con *collaborative filtering* por las ventajas que ofrecen [1], [2]. La segunda perspectiva que ha contribuido al desarrollo de modelos de recomendación viene del análisis predictivo, técnicas estadísticas que analizan los datos para clasificar o predecir la probabilidad de que un evento suceda. Los modelos predictivos han evolucionado desde modelos simples (regresión lineal o logística) a modelos que incorporan redes profundas (DL). El modelo DLRM surge de la combinación de ambas perspectivas, utilizando técnicas híbridas de sistemas de recomendación y modelos predictivos basados en DL para calcular la probabilidad de que un evento ocurra.

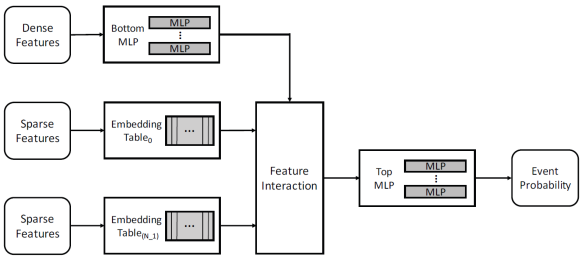


Fig. 1: Arquitectura del modelo DLRM [1].

Como se observa en la Figura 1, a alto nivel, el modelo hace uso de dos tipos diferentes de datos; *dense features* y *sparse features*; el modelo usa N *Embedding Tables* para procesar *sparse features*, que representan variables categóricas, nos referiremos a este componente como *EmbeddingBags*. Y un *multilayer perceptron*, MLP, para procesar las *dense features*, que son variables continuas. Posteriormente la salida de ambos algoritmos se combina en una fase denominada *Feature-Interactions* y se post-procesa en una *Top-MLP*, que determina la probabilidad de que un evento suceda.

En el ámbito matemático, *sparse* y *dense* son términos frecuentemente referidos al número de ceros contenidos en un vector o matriz. Por ejemplo, una matriz *sparse* estará compuesta mayoritariamente de ceros, mientras que en un vector dense, la mayoría de sus elementos serán no-nulos. Las entradas *dense* son procesadas por una *Bottom-MLP*, un *multilayer perceptron*, que se encarga de hacer una *GEneral Matrix Multiplication*, GEMM de las matrices de entrada. Como resultado se generará un vector *dense* que se enviará a la próxima fase, *Feature-interaction*. Por otra parte las entradas *sparse* serán procesadas por *EmbeddingBags*, un algoritmo que haciendo uso del formato CSR realizará operaciones GEMM sparse-

	Dense	Sparse	CSR	Reconfigurable
MAERI	✓	✗	✗	✓
TPU	✓	✗	✗	✗
SIGMA	✗	✓	✓	✓

Tabla I: Características de las arquitecturas aceleradoras.

dense, que explicaremos más adelante (Sección III), como por ejemplo, procesar productos escalares (*dot products* en inglés). Las matrices resultantes se agruparán formando una matriz dense que pasará a la siguiente fase.

En la fase denominada *Feature-Interactions*, se producen dos operaciones. Primero, la salida de la *Bottom-MLP*, un vector *dense*, se concatena con la salida del algoritmo *EmbeddingBag*, una matriz *dense*; es muy importante remarcar un aspecto, el vector dense ha de tener la misma dimensión (índices) que columnas tenga la matriz resultante con el fin de concatenarse (esto implicará un mayor o menor cómputo en la *Bottom-MLP*). A continuación, se realiza una *Matrix Factorization*. El resultado de esta fase, una matriz *dense*, se envía a una *Top-MLP* que devuelve un vector con la probabilidad de que un determinado evento suceda.

B. Aceleradores específicos para DLRM

Para acelerar el cómputo del modelo DLRM de manera eficiente, se necesitan arquitecturas aceleradoras especializadas tanto en el cómputo *dense* como en el cómputo *sparse*. Dada la reciente aparición del modelo DLRM, actualmente existen unos pocos trabajos relacionados que proponen plataformas de cómputo para aceleración de DLRM.

Por un lado, *Centaur* [2] es un acelerador híbrido compuesto por CPU y GPU integrado mediante un sistema de *multi-chiplet* (múltiples *chips* dentro de un mismo *chip*) (*CPU-Chiplet* + *FPGA-Chiplet*) para operaciones *sparse-dense*. En la primera parte del artículo ponen de manifiesto cómo de intensivas en términos de memoria son las *EmbeddingBags* y cómo de intensivas en cómputo son las MLPs. *Centaur* propone un subsistema de comunicación entre *chiplets* mediante un bus PCIe, donde los módulos encargados de realizar las operaciones *sparse-dense* han sido diseñados específicamente dentro de la *FPGA*.

Por otro lado, en otro trabajo presentado por *Facebook* [14], se pone de manifiesto la poca atención que han recibido las arquitecturas de los sistemas de recomendación, en especial la del modelo DLRM, a pesar de su importancia. Así, en este trabajo se presentan diferentes cargas de trabajo con el objetivo de demostrar la ineficiencia de los diferentes componentes. Estas cargas de trabajo se asemejan a cargas reales que podrían estar siendo ejecutadas en centros de computación actuales.

Un estudio de la literatura acerca de aceleradores para DNNs especializados en cómputo dense y sparse, nos lleva a diseños como el de la TPU, MAERI y SIGMA. Para procesamiento *dense*, serían adecuadas tanto la arquitectura TPU, como la de MAERI, mientras que para cómputo *sparse* está más especializada la arquitectura SIGMA, ya que permite procesar accesos a memoria en datos formateados en CSR (clave para el procesamiento de DLRM). Al ser tanto MAERI como SIGMA arquitecturas reconfigurables en tiempo de compilación, es decir, se pueden modificar las conexiones entre los distintos componentes de procesamiento (mediante pequeños conmutadores en sus redes de interconexión) para que el flujo de datos y procesamiento se adapte mejor a las características de cada etapa computacional (por ejemplo, una capa de una red neuronal), se van a usar MAERI y SIGMA para implementar un acelerador híbrido para acelerar la fase de inferencia de DLRM.

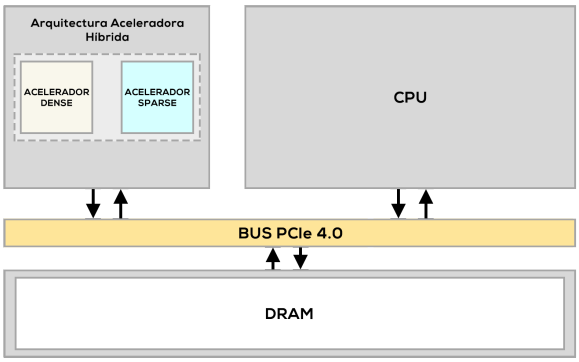


Fig. 2: Arquitectura híbrida propuesta.

La Tabla I muestra las características de cada una de las arquitecturas aceleradoras mencionadas.

III. Integración de DLRM en STONNE

La Figura 2 muestra los componentes básicos del acelerador híbrido que proponemos para acelerar la fase de inferencia de DLRM. Este acelerador híbrido integra dos aceleradores de uso específico para inferencia, uno para el cómputo dense (es decir, procesar las *MLPs*) y otro pare el cómputo *sparse* (es decir, procesar las *EmbeddingBags*). Como veremos, estos dos aceleradores se instanciarán en MAERI y SIGMA, respectivamente.

Para simular este acelerador y poder realizar la evaluación de DLRM, hemos partido de la herramienta de simulación STONNE[1], la cual nos permite modelar arquitecturas aceleradoras recientes tales como *MAERI*, *SIGMA* y la *TPU*. Además, nos permite hacer ejecuciones *end-to-end* de modelos de DL (e.g., las DNNs *Alexnet*, *Squeezenet*, *BERT*, etc). A pesar de esto, el simulador STONNE actualmente no está adaptado para implementar la arquitectura del acelerador híbrido propuesto.

Por otra parte, tenemos el modelo DLRM, desarrollado en *PyTorch*, que cuenta con 2 componentes fundamentales: las MLPs y las EmbeddingBags. STONNE no da soporte para DLRM, por lo que, en primer lugar tenemos que implementar las capas de DLRM necesarias y realizar una validación *end-to-end* (un benchmark) para comprobar su funcionamiento. Posteriormente, combinar la ejecución de ambos aceleradores para poder dar soporte completo a la ejecución del modelo DLRM.

[1]Repositorio - https://github.com/stonne-simulator/stonne

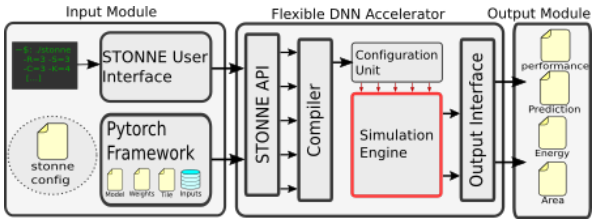


Fig. 3: El framework STONNE [16]

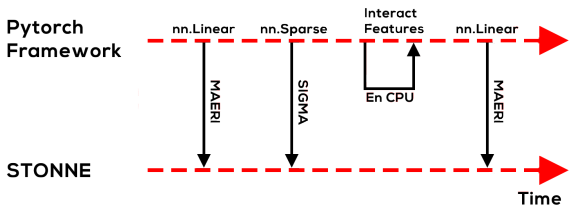


Fig. 4: Mapeo del modelo DLRM en la herramienta STONNE con las arquitecturas aceleradores MAERI y SIGMA.

STONNE permite conectarse a cualquier modelo de DL y actuar como acelerador (como si fuera una GPU o TPU real). En más detalle, la Figura 3 muestra los bloques principales de STONNE. Cuenta con un *front-end* escrito en *PyTorch*, que permite compilar código escrito en este lenguaje a código escrito en C++ para su posterior ejecución en el simulador. Mediante llamadas a esta API lograremos que el cómputo de las diferentes fases se realice en el simulador utilizando una arquitectura aceleradora (el *Simulation Engine*).

Como se explicó en la Sección II-A, el modelo DLRM cuenta con tres fases de ejecución bien identificadas: las MLPs, las *EmbeddingBags* y una fase denominada *Feature-Interaction*. Tras evaluar y comprobar el impacto en el cómputo que supone ejecutar esta última fase en CPU, se ha decidido no implementarla dentro del simulador por su baja influencia de cómputo. Como ya hemos mencionado, STONNE no da soporte para el modelo de recomendación DLRM, ya que la fase para el cómputo *sparse* (*EmbeddingBags*) no se encuentra implementada. Sin embargo, el simulador sí cuenta con la ejecución de MLPs de manera nativa (llamando a la API se puede realizar el cómputo *dense*), por lo que no va a ser necesario su integración.

Una vez llegados a este punto solo necesitaríamos integrar el cómputo *sparse*. Tenemos que dotar a STONNE de la funcionalidad para poder llamar a la API y que esta ejecute el algoritmo de las *EmbeddingBags*. Para este algoritmo necesitamos, primero, un conversor CSR que permita transformar los vectores CSR (*indices y offsets*) a un vector denominado *multi-hot encoding*. Y posteriormente una unidad de cómputo que se encargue de realizar una operación GEMM sparse del vector *multi-hot encoding* por la matriz de pesos de la *EmbeddingBag*. Este es el cómputo *sparse* del modelo.

Para realizar esta integración se ha duplicado la librería encargada de realizar las *EmbeddingBags*, llamada *nn.Sparse*, por *SimulatedSparse* (nos referiremos a ella como nn.Sparse por simplicidad). En el *forwarding* de la clase *EmbeddingBag* se ha llamado a la API de STONNE encargada de realizar esa operación GEMM sparse enviando como parámetros la matriz de pesos, el vector *multi-hot encoding* y otros parámetros necesarios del simulador. Una vez ambos componentes (MLPs y *EmbeddingBags*) han sido implementados, tendremos que especificar en STONNE una arquitectura a simular por STONNE (fichero *stonne config* de la Figura 3).

Existe una peculiaridad que es muy importante destacar de cara a la evaluación. En la *Top-MLP* además de las capas que especificaremos cuando confeccionemos los experimentos, el modelo DLRM realiza un cálculo interno; en base al número de *EmbeddingBags* y a la última capa de la *Bottom-MLP*, el modelo añadirá una capa extra con un determinado número de neuronas a la *Top-MLP*. Esta capa extra se ejecutará previamente al resto de las capas que nosotros le indiquemos.

IV. Evaluación

El simulador STONNE permite diferentes configuraciones para los aceleradores que simula. A continuación, pondremos de manifiesto qué metodología se ha llevado a cabo para realizar ciertos experimentos y posteriormente los resultados que hemos obtenido de integrar este modelo de recomendación dentro del simulador.

A. Metodología

Antes de presentar la metodología que vamos a seguir, es necesario contextualizar qué versión de *Python* y *frameworks* vamos a usar. En el caso de *Python* se ha utilizado la versión 3.8.0, el framework de *PyTorch* 1.7.0 y finalmente, para la gestión de librerías se ha usado *Anaconda* en la versión 4.9.2.

STONNE nos permite obtener un reporte de estadísticas de ejecución muy detallado tales como el número de ciclos de ejecución de cada componente de la arquitectura, el número de lecturas y de escrituras a las unidades de memoria internas (e.g., *Global Buffer*), el número de bits transmitidos por las redes dentro del chip, entre otras.

Como se ha mencionado en la Sección II-B, en este artículo se ha usado la arquitectura aceleradora *MAERI* [8] para el cómputo *dense* (MLPs) y la arquitectura aceleradora *SIGMA* [9] para el cómputo *sparse* de las *EmbeddingBags*. En la Figura 4 se puede ver la ejecución del modelo DLRM ya integrado completamente en STONNE. Además se representa la asignación de framework-acelerador para una ejecución *end-to-end* del modelo.

Para que el simulador STONNE simule dichas arquitecturas aceleradoras es necesario que se lo especifiquemos a través de un fichero de configuración (i.e., *stonne config*). Puesto que hemos usado las arquitecturas *MAERI* y *SIGMA*, cada una respectivamente para un componente diferente del modelo, hemos utilizado dos ficheros diferentes de configuración. En estos ficheros se especifican parámetros como el tamaño de los multiplicadores, el tipo de red de

	Número *PEs*	*Bandwidth* RD	*Bandwidth* RR	CM
MAERI	256	128	128	DENSE
SIGMA	128	128	128	SPARSE

Tabla II: Archivo de configuración para las arquitecturas. RR - Red de reducción. RD - Red de distribución. CM - Controlador de memoria

reducción, ancho de banda de la red de distribución y reducción, el controlador de memoria, etc.

En la Tabla II se muestran los parámetros más relevantes que hemos configurado para simular las arquitecturas *MAERI* y *SIGMA* de las que consta el acelerador híbrido propuesto. Como podemos observar, hemos escogido valores de parámetros similares a los considerados en sus trabajos. La diferencia principal entre ambas configuraciones se observa en el número de elementos de procesamiento (en inglés *Processing Elements* o *PEs*). En el caso de MAERI, hemos utilizado 256 PEs, mientras que en *SIGMA* este valor es reducido a la mitad, puesto que al explotar el *sparsity* se necesitan menos unidades de cómputo (i.e., se necesitan procesar menos operaciones en cada fase debido al gran número de operaciones que son innecesarias).

Además, puesto que el objetivo de este trabajo es el de realizar un análisis detallado del flujo de datos on-chip de la ejecución de DLRM sobre nuestras arquitecturas aceleradoras, para evitar entrar en resultados ligados a una jerarquía de memoria concreta entre el Global Buffer y la memoria principal, vamos a asumir un tamaño de Global Buffer ilimitado.

Para validar la integración del modelo en el simulador se han usado cargas de datos sintéticas similares a las propuestas en el artículo de *Centaur*. Los resultados de estas pruebas de validación han puesto de manifiesto la intensidad de cómputo de las MLPs y de memoria de las *EmbeddingBags*. Además, la mayor o menor intensidad de cómputo de las MLPs viene dada por el tamaño de *EmbeddingBags* que se use. Por este razonamiento hemos decidido mantener un valor fijo para ambas MLPs, teniendo en cuenta la capa extra que añade el modelo[2].

Para estudiar cargas de trabajo de DLRM similares a cargas reales necesitamos crear un modelo de recomendación con cierta envergadura en cuanto a número de *EmbeddingBags*, tamaño de las MLPs, etc. Para ello ha sido necesario investigar qué parámetros pueden modificarse y qué ejemplos de cargas de trabajo han sido propuestas por otros

[2]Parámetros *mlp-bot* = 128-64-64-32 y *mlp-top* = 128-64-1

Parámetro	Funcionalidad
embedding-size	Determina el número de *embeddings* que existirán y por cada embedding, cuántos índices(filas) tendrá.
mlp-{bot, top}	Indica cuántas capas y las neuronas por capa.
data-size	Determina el número de *lookups* que se realizarán en cada *EmbeddingBag*.
num-indices-lookup	Indica cuántos índices de la matriz de pesos de la *EmbeddingBag* se podrán seleccionar en cada *lookup*.

Tabla III: Parámetros configurables del modelo DLRM.

Grupo	Ejecución	Tam. *EmbeddingBag*
1	10-1M-20-100	128 *MBs*
	25-1M-20-100	128 *MBs*
	100-1M-20-100	128 *MBs*
2	10-1M-80-100	128 *MBs*
	25-1M-80-100	128 *MBs*
	50-1M-80-100	128 *MBs*
3	50-10M-20-100	1,28 *GBs*
	50-10M-80-100	1,28 *GBs*
	50-10M-20-1000	1,28 *GBs*
	50-10M-80-1000	1,28 *GBs*

Tabla IV: Cargas de trabajo sobre el modelo DLRM.

artículos [1], [2], [14]. En la Tabla III figuran los parámetros que vamos a modificar y el significado de cada uno de ellos.

Para referirnos a las diferentes cargas de trabajo dentro de cada grupo usaremos un sistema de nombres; el nombre de cada experimento vendrá definido por el número de *EmbeddingBags*, el número de índices por cada *EmbeddingBag*, su *data-size* y el *num-indices-lookup*.

Finalmente, en la Tabla IV presentamos las diferentes cargas de trabajo que vamos a ejecutar sobre el modelo DLRM. Se han usado tres grupos para clasificar los experimentos, el primero representa una carga de trabajo de "peso ligero", donde se usan 10, 25 y 50 *EmbeddingBags* respectivamente, manteniendo el parámetro *data-size* con un valor de 20. El segundo grupo representa una carga de "peso medio". El número de *EmbeddingBags* sigue siendo 10, 25 y 50 respectivamente, pero el parámetro *data-size* aumenta a 80. Esto tiene la finalidad de estudiar cómo afecta este parámetro a las lecturas y escrituras al *Global Buffer*. Finalmente, el tercer grupo representa una carga de trabajo "pesada", donde el número de *EmbeddingBags* es 50 en todos los casos, aumentamos el número de índices por *EmbeddingBag* y alteramos el valor *data-size*. Finalmente, en los dos últimos experimentos de este cuarto grupo modificamos el parámetro *num-indices-lookup* para estudiar su comportamiento sobre el modelo.

En los dos primeros grupos, el tamaño de cada *EmbeddingBag* es de 128 *MBs* y el de cada una de las MLPs es de 57 *KBs*. En el último grupo, las *EmbeddingBags* aumentan a 1,28 *GBs* mientras que las MLPs mantienen su tamaño inicial. Con este último experimento queremos poner a prueba el modelo DLRM, para ver cómo se comporta ante cargas de trabajo similares a las que podrían ejecutarse en un entorno real.

B. Resultados

A continuación presentamos los resultados obtenidos de los diferentes casos de estudio que hemos presentado anteriormente. En la Figura 5 se muestra el número de ciclos consumidos por cada una de las fases de ejecución, teniendo en cuenta el término MLPs como la suma de ciclos para la *Bottom-MLP* y la *Top-MLP*.

A primera vista, podemos observar como las MLPs

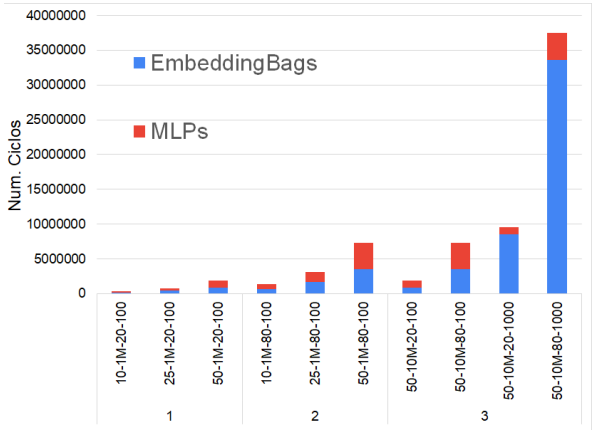


Fig. 5: Número de ciclos obtenidos.

(*Bottom* y *Top*) son intensivas en cómputo (ocupan la mayor parte de los ciclos). Si comparamos los ciclos de ejecución de las *EmbeddingBags* con los de las MLPs (experimentos del grupo 1 y 2), observamos cómo el cómputo *sparse* toma aproximadamente un 35-40 % del cómputo total, estos números se corresponden con los resultados esperados [2], [14]. Sin embargo, cuando aumentamos considerablemente el tamaño de las *EmbeddingBags*, al ser éstas tan intensivas en memoria, en los últimos experimentos (últimos dos experimentos del grupo 3) los ciclos de cómputo de este componente se llevan la mayor parte.

Además de obtener el número de ciclos para cada experimento, también se ha confeccionado una gráfica donde se muestra el número de escrituras y lecturas que se han realizado al *Global Buffer*. En la Figura 6 se muestra la gráfica. En este primer grupo de experimentos de carga ligera vamos a aumentar considerablemente el número de *EmbeddingBags*: comenzaremos por 10 y acabaremos con 50. Nos centraremos en observar la forma en la que el modelo se comporta conforme se aumenta este parámetro y mantenemos el resto estáticos.

Como se observa, las lecturas de la *Bottom-MLP* son las mismas para los tres experimentos. Esto se debe a los parámetros *mlp-bot* y *data-size*, cuyos valores no se modifican en este primer grupo, y por lo tanto, el número de lecturas para la *Bottom-MLP* va a ser el mismo en los tres casos. Con esto ponemos de manifiesto la relación que existe entre estos parámetros para determinar el número de lecturas de la *Bottom-MLP*.

Sin embargo, las lecturas de la *Top-MLP* sí varían en los tres experimentos. A pesar de especificar un tamaño fijo para la *Top-MLP*, si recordamos, el propio modelo DLRM realiza una operación en la que teniendo en cuenta el número de *EmbeddingBags* y el número de neuronas de la última capa de la *Bottom-MLP* añade una capa extra a la *Top-MLP*. Como en los tres casos aumentamos el número de *EmbeddingBags*, las neuronas de esta capa extra es diferente en los tres casos, como resultado, el número de lecturas es diferente.

Además, hay que tener en cuenta que el cómputo que ocurre en esta *Top-MLP* viene condicionado por el resultado de la fase anterior, *Feature-Interaction*. Dado que en esta fase se procesa la matriz resultante del algoritmo *EmbeddingBag*, al aumentar el número de *EmbeddingBags* o *data-size*, se aumenta el tamaño de la matriz resultante, por ende, las lecturas que se realizan en la *Top-MLP* también aumentan.

En el caso de las escrituras, dado que el mayor número se encuentra en las *EmbeddingBags*, se observa que estas se incrementan al aumentar el número de tablas. E concreto, al aumentar el número de *EmbeddingBags* se realizarán más operaciones GEMM sparse, como consecuencia, se generarán más escrituras.

En el caso de la *Bottom-MLP* y *Top-MLP*, ambos componentes realizan el mismo número de escrituras en los tres experimentos. A pesar de añadir una capa extra y diferente a la *Top-MLP* en cada experimento, esta realiza las mismas escrituras. Cuando presentemos el siguiente grupo, analizaremos los resultados y los compararemos con los obtenidos.

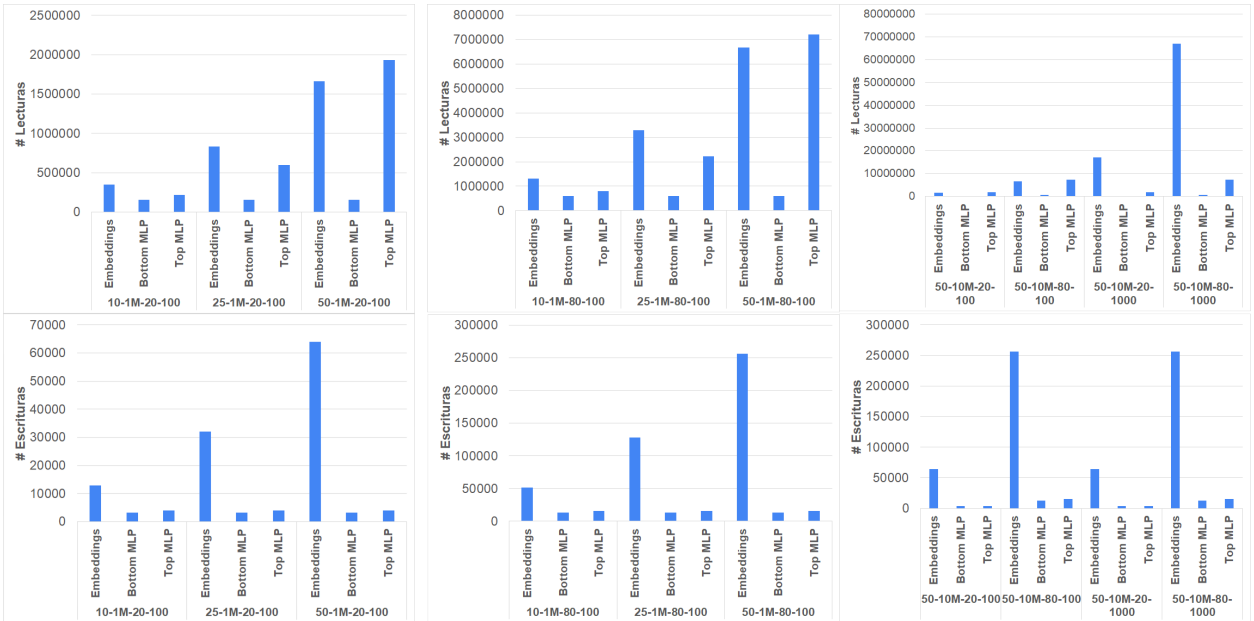
Teniendo en mente este primer grupo, vamos a introducir el grupo 2 con una carga de trabajo media. Se va a seguir la misma práctica que en el grupo 1, es decir vamos a ir aumentando el número de *EmbeddingBags*, comenzando por 10 y terminando con 50. Sin embargo, en vez de tener un valor de 20 para el parámetro *data-size*, vamos a aumentarlo a 80 (4×). Es decir, en vez de realizar 20 búsquedas por *EmbeddingBag*, vamos a realizar 80.

Si nos fijamos en la gráfica de las lecturas (gráfica superior de la Figura 6b) y la comparamos con la obtenida del grupo 1 (Figura 6a) vemos que comparte un aspecto similar. Las lecturas crecen de igual forma. En el caso de las lecturas de la *Bottom-MLP*, si aplicamos la teoría que hemos desarrollado antes, puesto que este vector tiene que tener la misma dimensión (índices) que la matriz resultante de las *EmbeddingBags* (columnas), al aumentar el número de búsquedas por *EmbeddingBag*, el vector dense que genera la *Bottom-MLP* ha de ser mayor. Concluyendo en que, al aumentar el parámetro *data-size* de 20 a 80 (4×) el número de lecturas de la *Bottom-MLP* aumenta en el mismo grado (4×) en los tres experimentos de este segundo grupo.

De este modo, corroboramos que las lecturas asociadas a este componente *Bottom-MLP* van en función de los parámetros *data-size* y *mlp-bot*.

Para las lecturas de las *EmbeddingBags*, al mantener el resto de parámetros estáticos y realizar 4 veces más búsquedas por *EmbeddingBag*, el número de lecturas aumenta notablemente y sigue un crecimiento similar a los resultados obtenidos en las lecturas de las *EmbeddingBags* del grupo 1 (Figura 6a).

Para el caso de la *Top-MLP* seguimos en la línea de lo ya comentado anteriormente. Al no modificar el número de *EmbeddingBags* en cada experimento y mantener el mismo tamaño de *Bottom-MLP*, esta capa extra que añade el modelo DLRM a la *Top-MLP* no se modifica y se aprecia un crecimiento similar al obtenido en el grupo 1.



(a) (b) (c)

Fig. 6: (a) L/E para el grupo 1. (b) L/E para el grupo 2. (c) L/E para el grupo 3. Todo en valor absoluto.

Para el caso de las escrituras, como el número de *EmbeddingBags* se mantiene entre ambos grupos, el número de escrituras va a ser el mismo, con la peculiaridad de que, al realizar 4 búsquedas más, el número de escrituras crece exactamente 4 veces más con respecto a los resultados obtenidos del grupo 1.

Como el valor de los parámetros *mlp-bpt* y *mlp-top* no se ha modificado y sabemos que las escrituras varían en función del valor de *data-size*, al haber aumentado este valor cuatro veces, las escrituras obtenidas en este grupo 2, son similares a las obtenidas en el grupo 1 multiplicadas por cuatro.

Estos experimentos nos sirven para darnos cuenta de cómo el parámetro *data-size* o búsquedas/*embedding* afecta al cómputo del modelo.

Hasta ahora hemos visto que le número de ciclos de ejecución que consumen las MLPs siempre solía ser mayor que el de las propias *EmbeddingBags*, y esto tiene sentido, ya que las MLPs son intensivas en cómputo. Sin embargo, hacemos especial atención a la necesidad de soportar grandes cargas de datos. Para ver cómo el modelo se comporta introducimos el último grupo.

En este tercer grupo introducimos una serie de experimentos con una carga de trabajo pesada, aproximando lo que podría considerarse como un *dataset* real.

Si nos fijamos en las dos primeras ejecuciones de este grupo, a pesar de haber aumentado 10 veces el número de índices de las *EmbeddingBags*, estas siguen sin suponer un gran porcentaje de los ciclos totales de cómputo (ocupan entorno al 46 % y 47 % de los ciclos totales). Este porcentaje de cómputo es el mismo que el obtenido en anteriores experimentos a pesar de aumentar las *EmbeddingBags* a 1,28 *GBs* (Figura 5).

En este punto introducimos el parámetro *num-indices-lookup*. Hasta ahora, para tablas de hasta 1 millón de índices este valor ha sido siempre de 100 unidades, lo que significa que por cada *lookup*/búsqueda que se realiza en la *EmbeddingBag*, como máximo se pueden coger 100 índices (0.0001 % de 1 millón de índices). Con el fin de realizar una mejor predicción, tendríamos que coger más índices, lo que se refleja en realizar más *lookups*. En los últimos dos experimentos se ha incrementado este número a 1000 y ha supuesto un impacto en el porcentaje de ciclos totales consumidos por las *EmbeddingBags* de un 89 % y 87 % respectivamente.

Basándonos en todo lo explicado anteriormente, vamos a explicar cómo este parámetro afecta al cómputo del modelo. Las lecturas de la *Bottom-MLP* y *Top-MLP* son iguales para el primer y tercer experimento, y para el segundo y cuarto, respectivamente (Figura 6c). Si nos fijamos, se mantienen los mismos parámetros que para el resto de experimentos (tercer caso de estudio del grupo 1 Figura 6a y grupo 2 Figura 6b), solo variamos el parámetro *num-indices-lookup*. Concluimos con las lecturas diciendo que este parámetro no afecta al número de lecturas en las MLPs.

Si nos fijamos en las *EmbeddingBags*, al aumentar el número de índices de las *EmbeddingBags* y el parámetro *num-indices-lookup*, todo esto acompañado de aumentar el parámetro *data-size*, número de *EmbeddingBags*, etc. se observa un gran aumento en el número de lecturas, aproximadamente en un factor de 10 (10×), puesto que hemos aumentado el parámetro *num-indices-lookup*. Este parámetro genera un fuerte impacto en el cómputo de las *EmbeddingBags* y el número de lecturas.

En el caso de las escrituras, seguimos en línea con lo comentado anteriormente. El número de escrituras de las MLPs viene determinado por el parámetro

data-size, *mlp-bot* y *mlp-top*. Puesto que no se modifican, se obtienen las mismas escrituras que en los grupos anteriores. Las escrituras de las MLPs para los experimentos 1 y 3 de este grupo, se corresponden con las obtenidas en el grupo 1 (tercer caso de estudio Figura 6a); y los experimentos 2 y 4 con las obtenidas en el grupo 2 (tercer caso de estudio Figura 6b).

Para las escrituras de las *EmbeddingBags* ocurre algo peculiar. Las escrituras en los experimentos 1 y 3 de este grupo, se corresponden con las obtenidas en el experimento 3 del grupo 1 (Figura 6a), mismo valor del *data-size* (20) y mismo número de *EmbeddingBags*(50). Por otra parte, las escrituras obtenidas en los experimentos 2 y 4 (los más intensivos en cómputo), se observa como las escrituras se corresponden con las que se han obtenido en el experimento 3 del grupo 2 (Figura 6b) (mismo *data-size* y mismo número de *EmbeddingBags*).

C. Resumen del Análisis de Resultados

A continuación presentamos un resumen de las conclusiones obtenidas tras haber realizado los experimentos:

- Si comparamos el tiempo total empleado por los diferentes experimentos, el componente más costoso, donde más tiempo se ha empleado, ha sido en el procesamiento de las *EmbeddingBags*.
- En el caso de las lecturas, sabemos que el mayor o menor número de lecturas en las *EmbeddingBags* viene condicionado por el número de *EmbeddingBags*, de los índices de cada *EmbeddingBag*, del parámetro *data-size* y del parámetro *num-indices-lookup*.
- En el caso de la *Bottom-MLP*, las lecturas vienen determinadas por el parámetro *data-size* y por el tamaño del parámetro *mlp-bot*, que determina el número de capas y las neuronas por capa.
- Para la *Top-MLP*, recordemos que el modelo añade una capa extra previa a las que añadimos nosotros. De esta manera, las lecturas vienen condicionadas por el número de *EmbeddingBags*, las neuronas de la última capa de la *Bot-MLP* y el parámetro *data-size*. Obviamente, dejando estos parámetros estáticos y modificando el valor del parámetro *mlp-top*, también se verá reflejado un aumento o disminución en las lecturas.
- El número de escrituras en las *EmbeddingBags* viene determinado por el número de *EmbeddingBags* que existan y por el parámetro *data-size*. El aumento del parámetro *num-indices-lookup* ha demostrado no suponer ningún tipo de impacto en las escrituras.
- Para el caso de las MLPs es mas sencillo, puesto que las escrituras vienen determinadas por el número de capas, es decir, *mlp-*{*top,bot*} y el parámetro *data-size*.

V. Conclusiones

En este trabajo hemos realizado una evaluación de la ejecución del proceso de inferencia de DLRM sobre un acelerador híbrido propuesto simulado con STONNE. Este acelerador híbrido está compuesto por dos aceleradores, uno para cómputo *sparse* y otro para cómputo *dense*, de uso específico para inferencia. Hemos escogido las arquitecturas MAERI y SIGMA, ya que son reconfigurables y ofrecen notables ventajas respecto a otras arquitecturas, como podría ser la TPU.

De los resultados obtenidos podemos observar cómo el tiempo de ejecución aumenta notablemente al aumentar el tamaño y número de *EmbeddingBags*, así como el *data-size*. Por otra parte, las lecturas y escrituras de los diferentes componentes del modelo están correlacionadas con el tamaño de las *EmbeddingBags*. Como ya hemos visto en los experimentos, el mayor o menor número de ciclos, escrituras o lecturas durante el procesamiento viene dado por el tamaño de *EmbeddingBags*, que es el componente responsable del cómputo *sparse*.

Gracias a los resultados de la evaluación realizada, como trabajo futuro se va a mejorar la arquitectura del acelerador híbrido propuesto implementando una jerarquía de memoria que soporte de forma eficiente la inmensa cantidad de lecturas y escrituras de memoria que el modelo DLRM necesita durante su procesamiento *sparse*.

Agradecimientos

Trabajo financiado por RTI2018-098156-B-C53 (MCIU/AEI/FEDER,UE), NSF OAC 1909900 y US Department of Energy ARIAA co-design center. Francisco Muñoz-Martínez ha sido financiado mediante la beca 20749/FPI/18 de Fundación Séneca, Agencia de Ciencia y Tecnología de la Región de Murcia.

Referencias

[1] Dheevatsa Mudigere Maxim Naumov et al., "Deep learning recommendation model for personalization and recommendation systems," *CoRR*, vol. abs/1906.00091, 2019.

[2] Taehun Kim Ranggi Hwang et al., "Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations," *CoRR*, vol. abs/2005.05968, 2020.

[3] Nishigandha Karbhari et al., "Recommendation system using content filtering: A case study for college campus placement," in *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, 2017, pp. 963–965.

[4] Ben Schafer et al., "Collaborative filtering recommender systems," 01 2007.

[5] Robert M Bell and Yehuda Koren, "Improved neighborhood-based collaborative filtering," in *KDD cup and workshop at the 13th ACM SIGKDD international conference on knowledge discovery and data mining*. Citeseer, 2007, pp. 7–14.

[6] Yehuda Koren, "Factorization meets the neighborhood: A multifaceted collaborative filtering model," New York, NY, USA, 2008, KDD '08, p. 426–434, Association for Computing Machinery.

[7] Norman P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *44th Int'l Symp. on Computer Architecture*, 2017.

[8] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna, "MAERI: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," *Int'l*

Conf. on Architectural Support for Programming Languages and Operating Systems, Mar. 2018.

[9] Eric Qin et al., "SIGMA: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," *Int'l Symp. on High-Performance Computer Architecture*, Mar. 2020.

[10] Vahideh Akhlaghi, Amir Yazdanbakhsh, Kambiz Samadi, Rajesh K. Gupta, and Hadi Esmaeilzadeh, "SnaPEA: Predictive early activation for reducing computation in deep convolutional neural networks," *International Symposium on Computer Architecture (ISCA)*, pp. 662–673, June 2018.

[11] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292 – 308, June 2019.

[12] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally, "SCNN: An accelerator for compressed-sparse convolutional neural networks," *International Symposium on Computer Architecture (ISCA)*, pp. 27–40, June 2017.

[13] Malay Haldar et al., "Applying deep learning to airbnb search," New York, NY, USA, 2019, KDD '19, p. 1927–1935, Association for Computing Machinery.

[14] Xiaodong Wang Udit Gupta et al., "The architectural implications of facebook's dnn-based personalized recommendation," *CoRR*, vol. abs/1906.03109, 2019.

[15] Mingyu Yan, Lei Deng, Xing Hu, Ling Liang, Yujing Feng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie, "Hygcn: A gcn accelerator with hybrid architecture," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 15–29.

[16] Francisco Muñoz Matrínez, José L. Abellán, Manuel E. Acacio, and Tushar Krishna, "Stonne: A detailed architectural simulator for flexible neural network accelerators," *arXiv preprint arXiv:2006.07137v1*, June 2020.