International Journal of
HIGH PERFORMANCE
COMPUTING APPLICATIONS

Special Issue Paper

The International Journal of High
Performance Computing Applications
1–14
© The Author(s) 2020
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/1094342020953196
journals.sagepub.com/home/hpc
SAGE

Convolutional neural nets for estimating the run time and energy consumption of the sparse matrix-vector product

Maria Barreda, Manuel F Dolz and M Asunción Castaño

Abstract

Modeling the performance and energy consumption of the sparse matrix-vector product (SpMV) is essential to perform off-line analysis and, for example, choose a target computer architecture that delivers the best performance-energy consumption ratio. However, this task is especially complex given the memory-bounded nature and irregular memory accesses of the SpMV, mainly dictated by the input sparse matrix. In this paper, we propose a Machine Learning (ML)-driven approach that leverages Convolutional Neural Networks (CNNs) to provide accurate estimations of the performance and energy consumption of the SpMV kernel. The proposed CNN-based models use a blockwise approach to make the CNN architecture independent of the matrix size. These models are trained to estimate execution time as well as total, package, and DRAM energy consumption at different processor frequencies. The experimental results reveal that the overall relative error ranges between 0.5% and 14%, while at matrix level is not superior to 10%. To demonstrate the applicability and accuracy of the SpMV CNN-based models, this study is complemented with an ad-hoc time-energy model for the PageRank algorithm, a popular algorithm for web information retrieval used by search engines, which internally realizes the SpMV kernel.

Keywords

Sparse matrix-vector multiplication (SpMV), performance modeling, energy modeling, supervised learning, convolutional neural networks (CNN), PageRank algorithm

1 Introduction

Sparse matrices often appear in the solution of large linear systems as part of complex simulations, such as, for instance, in finite element methods, economic modeling and search methodologies for information retrieval. In these simulations, the Sparse Matrix-Vector (SpMV) product plays a fundamental role for the iterative solution of sparse linear systems, as it frequently dictates the execution time and energy consumption of the whole algorithm. Indeed, the SpMV operation has been characterized as one of the most important computational kernels in science and engineering for the last decade (Gkountouvas et al., 2013).

Several algorithmic characteristics render the study of the SpMV kernel challenging: concretely, its irregular and indirect memory accesses, which negatively impact the cache spatial and temporal localities. These effects are, in turn, translated into a very low flop per byte ratio, leading to bottlenecks in the memory access (Elafrou et al., 2017). This not only adds significant overhead in the execution time but also increases energy consumption. In fact, fetching data from the DRAM for an operation consumes an amount of energy which is orders of magnitude higher than the computation itself. Thus, as computation becomes more energy-efficient, the cost of data movement gradually becomes a more relevant issue (Llopis et al., 2016). To progress in this direction, time-energy models for memory-bounded, power-hungry algorithms, such as the SpMV kernel, are key pieces to enable the design of a new generation of energy-efficient memory architectures.

Some of the aspects that dictate the poor performance of the SpMV kernel are the non-zero sparsity pattern and the row-density of the sparse matrix involved in the computation.[1] These elements, together with the sparse matrix storage format, such as Compressed Sparse Row (CSR), Coordinate list (COO), ELLPACK, etc. determine the sequence of memory accesses and, consequently, the succession of time-costly cache misses. Over the last few years, a considerable amount of research has focused on

Departament d'Enginyeria i Ciència dels Computadors, Universitat Jaume I de Castelló, Spain

Corresponding author:
Manuel F Dolz, Departament d'Enginyeria i Ciència dels Computadors, Universitat Jaume I, Avda. Sos Baynat s/n, Castellón de la Plana, Castellón 12071, Spain.
Email: dolzm@uji.es

designing performance/energy models for the SpMV kernel (see Elafrou et al., 2017; Malossi et al., 2014 and the references therein). These models often rely on architectural memory-parameters (such as memory bandwidth or cache replacement policies) and the sparse matrix features to estimate the theoretical throughput (operations per second) at which the SpMV product can be realized. Similarly, energy models use average net-energy values per non-zero element which, in general, only provide estimations of the total energy consumption drawn by the processor. Furthermore, all of them require both a deep understanding of the processor architecture as well as a detailed analysis of the SpMV implementation (Li et al., 2015).

Machine learning (ML), a subset of Artificial Intelligence algorithms, is an alternative to analytical models that can automatically derive mathematical models from sample data with minimum human intervention. Neural Networks (NNs), in particular, have shown the ability to approximate complex nonlinear functions. Specifically, Convolutional Neural Networks (CNNs) (Gu et al., 2018) may provide a powerful means to capture spatial and temporal dependencies using abstract representations of the sparse matrices involved in the SpMV through a set of convolutional filters. In this paper, we extend the work in (Barreda et al., 2020c) with the design of a more accurate execution time model and a new energy consumption model using CNNs as base algorithms. In particular, our paper makes the following contributions:

- We leverage CNNs to model the execution time and energy consumption of the SpMV on an Intel Xeon Haswell core using CSR (Eijkhout and Pozo, 1994) as the storage format.[2] The approach carries over to any other specialized sparse matrix storage format.
- We study the estimations of these metrics considering different operating frequencies of the Intel Xeon Haswell core.
- We leverage the model migration technique presented in (Barreda et al., 2020c) for the different energy metrics and frequencies studied.
- We propose a blockwise realization to make the CNN model architecture independent of the sparse matrix dimension input blocks.
- We introduce a normalization method for each input block in order to facilitate the training of the CNNs.
- We evaluate the accuracy and demonstrate the robustness of the CNN-based models using a representative subset of real applications from the SuiteSparse Matrix collection (Davis and Hu, 2011).
- We derive analytical models for estimating the run time and energy consumption of the PageRank algorithm. The presented piecewise models combine our CNN-based models to infer the cost of the SpMV operation and the well-known roofline model to estimate the time and energy for the remaining kernels present in PageRank.

The most obvious application for our off-line time-energy estimators is that, given trained models for a variety of processor architectures, selecting the best option (processor and configuration, e.g. operation frequency) does not require direct access to the target platforms. In particular, inference can be run off-line on the (trained) models, on a single architecture different from those which the CNNs model. Also, being able to estimate the execution cost of an irregular and challenging operation such as the SpMV paves the way toward applying similar ML-driven techniques to modeling the cost of memory accesses for more complex numerical kernels or even general-purpose applications.

The rest of this paper is organized as follows. Section 3 reviews some basic concepts about the SpMV kernel and CNNs. Section 2 revisits a few other works related to performance and energy modeling and/or linear algebra operations using NNs. Section 4 describes the strategy to accommodate the CSR format as a valid input for the CNN models and details the architecture of the CNN. Section 5 evaluates the training process of the proposed models tuned with hyperparameter optimization and analyzes the accuracy attained by the networks for the SpMV. Section 6 presents two piece-models that combine the CNN-based predictors with the roofline strategy to estimate the execution time and energy consumption of the PageRank algorithm. Finally, Section 7 offers a few concluding remarks and summarizes future research lines.

2 Related work

Artificial NNs were introduced in the 1950s. They were widely used in the 1980s although their abilities were computationally limited. In the last years, the availability of big data and the increasing computational power of modern accelerator architectures have allowed deeper NNs, resulting in Deep Learning (DL). Besides, tools like Keras (Chollet, 2015) and TensorFlow (Abadi et al., 2015) have contributed to making neural networks and DL more accessible. CNNs (Gu et al., 2018), a class of deep NNs with at least one convolutional layer, have shown excellent accuracy on many ML-related areas (Gu et al., 2018; LeCun et al., 2015; Schmidhuber, 2015). However, the potential of DL is still largely unexplored in linear algebra. In particular, only a few works related to the SpMV operation and/or performance, power and energy modeling have been previously approached using NNs.

Götz and Anzt (2018) leverage 2D CNNs to detect strongly connected blocks of sparse matrices in order to derive block-Jacobi preconditioners and accelerate the iterative solution of the corresponding linear system. Similarly, both Zhao et al. (2018) and Cui et al. (2018) feed matrix sparsity images into CNNs which are trained to select the most adequate matrix storage format of the SpMV operation. Zhao et al. also employ transfer learning to alleviate cross-architecture migration of the CNN-based models. Nisa et al. (2018) predict the best storage format for the

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 \\ 0 & 4 & 5 & 0 \\ 0 & 0 & 0 & 6 \end{pmatrix} \qquad \begin{array}{l} A_{vval[6]} = \{1, 2, 3, 4, 5, 6\} \\ A_{vptr[5]} = \{0, 2, 3, 5, 6\} \\ A_{vpos[6]} = \{0, 3, 2, 1, 2, 3\} \end{array}$$

Figure 1. Example of a sparse matrix *A* stored in CSR format as three vectors: *vval*, *vptr* and *vpos*. Indices are numbered starting at 0.

SpMV operation. However, instead of using 2D images as in the previous works, sparsity metrics are used as inputs for a Multilayer Perceptron (MLP).

Concerning performance and power modeling, Tiwari et al. (2012) use kernel-specific compiler-based optimization parameters and hardware tunables to train MLPs and estimate the performance, power, and energy usage of several computational kernels. Both Benatia et al. (2016) and Nisa et al. (2018) leverage MLPs to predict the GPU performance of the SpMV kernel using different matrix storage formats on GPU architectures. Song et al. (2013) derive power models for GPU-based systems by training MLPs for distinct CUDA kernels using hardware performance counters. Endrei et al. (2019) use a MLP to predict Pareto frontiers in order to analyze the trade-offs between performance and energy usage, for several parallel kernels and complex application workloads.

The aforementioned performance and power modeling techniques are different to the approach proposed in this work in four main aspects: *i)* we use CNNs instead of MLPs, so that spatial features of sparse matrices can be automatically captured; *ii)* we feed the sparse matrix structure directly into the CNN instead of using sparse matrix metrics related to such structure or hardware performance counters; *iii)* our blockwise methodology allows creating large datasets to train the CNNs in order to estimate the run time and energy consumption of the SpMV operation; and *iv)* the use of CNNs for estimating the aforementioned metrics at different frequencies allows developers to select the computer architecture with the best performance-energy consumption ratio.

3 Background

3.1 The sparse matrix-vector product

The SpMV kernel computes the sparse-matrix-vector product $y = Ax$, where A is an input sparse matrix of size $m \times n$ (with *nnz* non-zero elements); x is an input dense vector, of size n; and y is an output dense vector, of size m. Usually, the elements of A are stored using the Compressed Sparse Row (CSR) format, as this provides a flexible, memory-efficient, and architecture-agnostic solution. This format stores the matrix using three arrays that contain, respectively, the non-zero values (*vval*), the row pointers (*vptr*), and the column index of each element (*vpos*). Figure 1 shows a simple example of a 4 × 4 matrix stored using the CSR format.

Algorithm 1. Realization of the SpMV algorithm using the CSR format.

```
Require: $A \to m \times n, x \to n, y \to m$
for $i = 0, 1, \ldots, n-1$ do
    for $j = A_{vptr[i]}, A_{vptr[i]+1}, \ldots, A_{vptr[i+1]-1}$ do
        $y[i] := y[i] + A_{vval[j]} \cdot x[A_{vpos[j]}]$
    end for
end for
```

Algorithm 1 presents the SpMV implementation. Specifically, the external loop (indexed by i) iterates over the matrix rows, whereas the internal loop (indexed by j) progresses through the entries of each row, using the *vptr* and *vpos* vectors to recover the corresponding index to access x. Finally, the value at the coordinate (i,j) of A is retrieved using the *vval* array. As previously mentioned, the irregular memory accesses of this implementation occur while retrieving the elements of the x vector, which are indirectly accessed through the *vpos* array.

3.2 Convolutional neural networks

CNNs usually consist of a collection of convolutional (Conv) layers followed by a reduced number of fully-connected (FC) layers (Gu et al., 2018). The result of convolving the filter over the entire Conv layer is called feature map. Different filters can be applied in a Conv layer to obtain multiple feature maps. The resulting activations in the feature maps are then passed through a nonlinear function, such as the Rectified Linear Unit (ReLU), which has been reported to achieve fast training in supervised learning of DL networks (Glorot et al., 2011).

Pooling layers semantically merge similar information by creating downsampled and summarized versions of the features detected by the Conv layers (Boureau et al., 2010). Consequently, they reduce the number of connections between Conv layers and the computational power required to process the data. Typical pooling functions are maximum and average, though other more complex functions, such as gated max-average pooling, are also possible (Lee et al., 2015).

Dropout layers are frequently included in CNNs to improve the learning process and to prevent network overfitting. Dropout is a regularization technique that ignores randomly-selected neurons and their corresponding weights during training. This prevents the co-dependency among neurons which in turn leads to a better generalization on new input data.

In the FC layers, each neuron is connected to all neurons of the next layer. FC layers integrate all the high-level features extracted by the previous Conv layers to produce the final outputs. Depending on whether the CNN is a realization of a classification problem or a regression model, the last layer may contain as many neurons as classes or a single one. In the former case, the neurons are activated via nonlinear functions (e.g., sigmoid, softmax, or

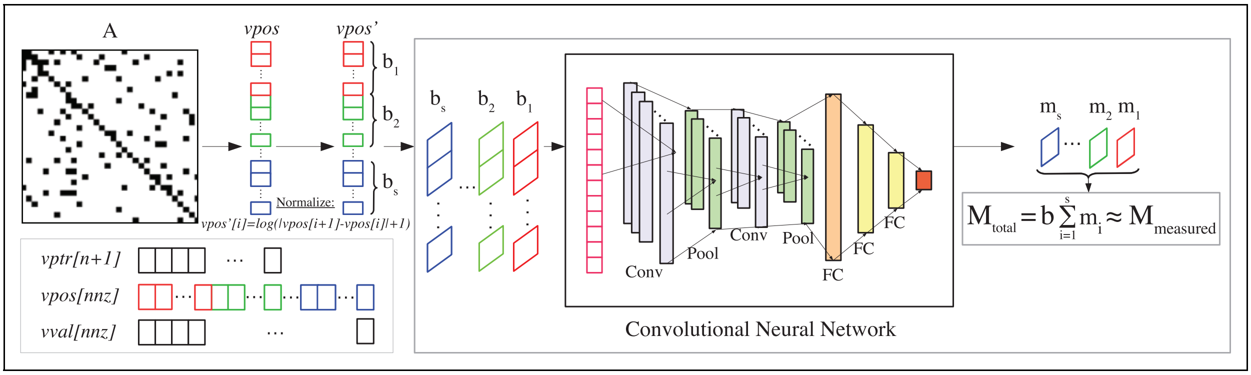
A
vpos
vpos'
b1
b2
bs
Normalize:
vpos'[i]=log(|vpos[i+1]-vpos[i]|+1)
vptr[n+1]
vpos[nnz]
vval[nnz]
bs
b2
b1
Conv
Pool
Conv
Pool
FC
FC
FC
Convolutional Neural Network
ms
m2
m1
$M_{total} = b\sum_{i=1}^{s} m_i \approx M_{measured}$

Figure 2. Workflow for modeling the SpMV performance and energy consumption. The term "M" in the expression denotes any of the selected metrics: execution time, total, package or DRAM energy consumption.

ReLU), while in the latter a linear function activates the single neuron.

4 Modeling the SpMV via CNNs

In this section, we detail the methodology employed to estimate the performance and energy consumption of the SpMV kernel using CNNs. In our approach, we generate individual CNN models[3] for the estimated metrics, i.e. execution time and energy. Also, considering that our target core processor can work at different frequencies and that the energy consumption (measured via the Intel RAPL) can be split into three different components (total, package, and DRAM), distinct CNN models are considered for each evaluated metric (execution time, and total, package, and DRAM energy consumption) and frequency.

4.1 Methodology

The SpMV product is a memory-bound operation in which the irregular sparsity pattern of matrix A dictates the non-sequential accesses to the x vector and, therefore, the volume of L2/L3 cache misses and DRAM memory accesses. In this sense, the *vpos* array can be considered as a key element to understand the different arithmetic-to-memory access intensities that, in turn, allow to estimate both the performance and energy consumption of the SpMV kernel.

With this idea in mind, we design a CNN that will learn from the information contained in *vpos*. Using this approach, the convolutional filters capture meaningful features in this array (e.g., patterns of distances between non-zero entries), which may yield useful estimations of the SpMV performance and energy consumption. For this purpose, we preprocess the *vpos* array in the following way. First, we compute the differences between consecutive elements of *vpos*, instead of directly passing the column indices to the CNN. This eases the learning process, as we guide the network to directly focus on the column distances. In a second step, we normalize the input data via the log function in order to narrow the range of values that the column distances may take and to facilitate the network learning. In sum, each element of the *vpos* array is transformed into $vpos'[i] = \log(|vpos[i+1] - vpos[i]| + 1)$.

Figure 2 resumes the workflow adopted in the modeling task. Given that the sparse matrices may present large variations on their size and number of non-zero entries (*nnz*), we split the transformed $vpos'$ array into chunks (or blocks) of size b which can then be sequentially fed to a CNN design with a constant number of inputs. With this approach, the CNNs can be used uniformly to predict individual execution times and energy consumptions of equal-sized blocks, which may belong to any sparse matrix, regardless of its size and number of non-zeros. Thus, considering that t_i and e_i are, respectively, the execution time and energy consumption per non-zero element of the i-th block of A, the estimated total execution time and energy consumption for this matrix can be calculated as the aggregation of the time/energy-per-element for the $\lceil nnz/b = s \rceil$ blocks multiplied by b; that is, $T_{total} \approx b\sum_{i=1}^{s} t_i$ and $E_{total} \approx b\sum_{i=1}^{s} e_i$. Note that the latter expression applies for the three energy components studied, i.e. total, package, and DRAM. The proposed design iteratively supplies each block of the transformed $vpos'$ to the trained CNN so that the outputs obtained by the network correspond to predictions of the partial execution time/energy (per element) of this block, which can then be added to obtain the total execution time or energy consumption of the SpMV for the matrix A.

The strategy of partitioning the $vpos'$ array into blocks forces us to implement a blockwise version of the classic CSR-based SpMV Algorithm 1 to generate the training dataset for the CNNs, in which each block of $vpos'$ is labeled with its corresponding ratios of execution time and energy consumption (total, package, and DRAM) per non-zero element. In a previous work (Barreda et al., 2020c), we analyzed the impact of the block size in the time predictions and concluded that the proposed network architectures deliver accurate results for small block sizes. The reason is that, in general, small blocks reflect a small set of sparsity patterns which, in turn, can be better captured by the CNN filters. Consequently, having small block sizes increases the execution time and energy consumption variability among blocks, so that the predictions vary in a wider interval. However, this is only true to a certain extent. In particular, using block sizes that are too small (below 250) may negatively affect their execution time due to cache data locality effects. On the other hand, working with large block sizes may lead to homogeneous execution time/energy labels, preventing the CNN to learn important sparsity features comprised in a single block. Thus, taking into account the lessons learned in that study, in this work we use a fixed block size $b = 250$. For this configuration, if the number of *nnz* entries is not multiple of b, the remaining block of size $b^{'} < b$ is discarded. In our case, this is safe enough as the smallest value of *nnz* for the selected sparse matrices is always higher than 1 million. In consequence, with $b = 250$, only one block among 4,000 is discarded.

4.2 Network architecture

Considering our blockwise strategy, we aim to design a CNN architecture that offers accurate estimates of the evaluated metrics for the SpMV operation executed at different processor frequencies. Our goal is thus to design a simple network architecture, inspired by the structure of well-known CNNs, such as AlexNet, LeNet or VGG, and demonstrate that it can also deliver accurate execution time/energy estimations of the SpMV kernel.

For the network architecture, we designed a one-dimensional (1D) CNN-based regression model, where the activation of the single output neuron is the estimated value for the studied metric per non-zero element. In our previous study (Barreda et al., 2020c), we tested different sequences of convolutional and pooling layers, achieving similar performances in all of them. Consequently, in this work, we adopted the simplest combination, which consists of a sequence of one convolutional layer followed by a max-pooling layer, which is repeated twice. For the second part of the network, we appended some FC layers to combine the features captured in the previous convolutional stages. Also, we included a dropout layer between any two consecutive FC layers. Finally, the last FC layer generates the estimated output.

It is important to remark that some configuration parameters of the networks are established as hyperparameters, e.g. the number of stacked FC layers at the end of the net, the number of neurons per FC layer, and the number of filters in the convolutional layers. The values adopted for these parameters are set by a hyperparameter optimization tool before the training stage.

Given the high number of combinations among the selected metrics and the frequencies, obtaining adequate values for the hyperparameters becomes a time-consuming process. Therefore, we only calculate the hyperparameters for the models estimating the execution time and total energy consumption at the highest processor frequency. Taking into account the distinct nature and units of the modeled metrics, it is reasonable that the models for the rest of frequencies and metrics (package and DRAM energy) can inherit the hyperparameters of the corresponding metric, i.e. execution time or energy consumption.

5 Experimental evaluation

In this section, we describe *i)* the generation of the training and testing datasets; *ii)* the hyperparameter optimization and training process; and *iii)* the testing evaluation with a set of sparse matrices through relative mean errors of the SpMV for the selected metrics and frequencies. To carry out these tasks of the experimental evaluation, we have employed the following hardware and software components:

- Hardware:
  - The compute node where the networks were trained consisted of two Intel Xeon E5-2698, with a total of 40 cores clocked at 2.20 GHz, and four NVIDIA Tesla P100 GPUs with 16 GB of DRAM at 1.48 GHz interconnected via NVLink.
  - The execution time and energy consumption data corresponding to the SpMV operation were obtained on an Intel Xeon E5-2630 core running at the selected frequencies ($\{1.2, 1.6, 2.0, 2.4\}$ GHz). These frequencies were set via the `user-space` governor from the `acpi-cpufreq` driver using the `cpufreq-set` Linux utility. The energy consumption was measured via the Intel RAPL interface and gathered at three different levels (total, package, and DRAM, where total = package + DRAM) for this specific processor.
- *Software:* The framework for building the CNNs was Keras v2.2.4 (Chollet, 2015) on top of TensorFlow r1.10 (Abadi et al., 2015). Moreover, we leveraged Hyperas v0.4.1 (Pumperla, 2017), a wrapper around Hyperopt (Bergstra et al., 2013) that implements an algorithm for hyperparameter optimization targeted to Keras-based models. Finally, the ad-hoc SpMV benchmark was implemented in C and compiled with gcc 5.3.0 with the `-march=native -O1` flags.

The training and evaluation workflow proceeds as follows. First, we build the training and testing datasets, where each input block is tagged with the corresponding measured values related to the selected metrics by executing the SpMV operation on the target processor core at the considered frequencies. Next, we obtain the tuned versions of the models by performing the hyperparameter search. Afterward, we train the models using the aforementioned dataset. Finally, for each sparse matrix in the testing dataset, we apply the trained models to estimate the total execution time and the energy consumptions of the SpMV operation (inference) and compute the relative errors concerning the metrics and frequencies.

5.1 Obtaining the dataset

The training and testing datasets were obtained by realizing the blockwise SpMV operation at the CPU frequencies for

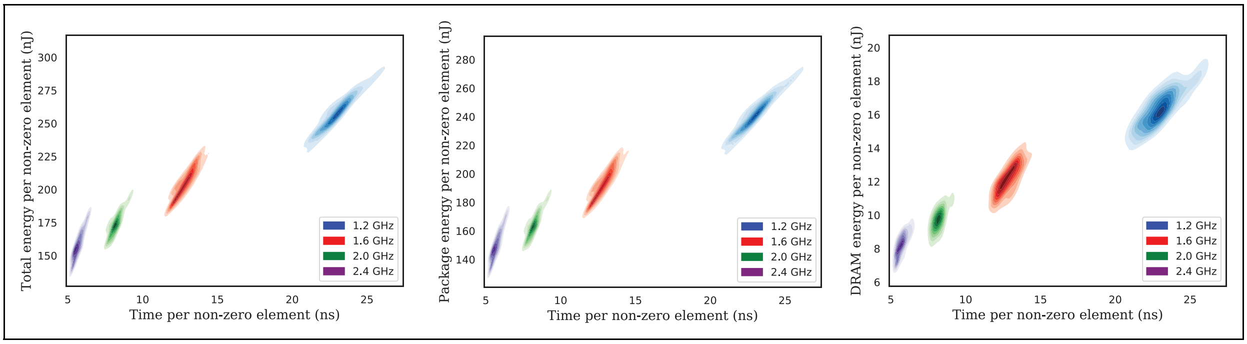

Figure 3. Dispersion plots for execution time vs total, package, and DRAM energies at different frequencies.

Table 1. Execution time and total/package/DRAM energy consumption ranges, means (µ), and standard deviations (σ) for the dataset blocks of size $b = 250$.

Frequency (GHz)	Execution time (µs)			Total energy (µJ)			Package energy (µJ)			DRAM energy (µJ)		
	Range	µ	σ	Range	µ	σ	Range	µ	σ	Range	µ	σ
1.2	[5.33, 8.29]	5.71	0.28	[57.32, 94.25]	64.11	3.35	[53.85, 88.27]	60.05	3.13	[2.94, 7.36]	4.06	0.25
1.6	[3.00, 4.68]	3.22	0.16	[44.85, 74.63]	51.35	2.78	[42.45, 69.70]	48.29	2.61	[1.99, 6.23]	3.05	0.21
2.0	[1.92, 2.97]	2.05	0.10	[37.56, 63.39]	43.60	2.46	[35.53, 59.95]	41.18	2.32	[1.48, 5.05]	2.44	0.18
2.4	[1.33, 2.06]	1.42	0.07	[34.52, 58.40]	39.58	2.51	[32.88, 55.39]	37.54	2.39	[1.14, 4.29]	2.03	0.16

the studied sparse matrices, as detailed in Section 4.1, while measuring the metrics per non-zero element in each $vpos'$ block. To this end, we selected 186 sparse matrices from the SuiteSparse Matrix Collection (Davis and Hu, 2011) with a number of non-zeros ranging between 1 M and 10 M. The time taken to preprocess the sparse matrices, realize the blockwise SpMV and build the dataset took about a week. From these matrices, 80% was dedicated for training, while the remaining 20% was reserved for testing. Similarly, 80% of the training dataset was utilized only for training, and the remaining 20% was used for validation, to guide the training process. In Barreda et al. (2020c) we showed that the trained models for the time metric using this dataset provide an appropriate generalization power. Using too much training data could lead to a model which overfits the problem. Also note that a concrete sparse matrix with nnz non-zero entries yields $\lceil \frac{nnz}{b} \rceil = \lceil \frac{nnz}{250} \rceil$ blocks that are part of the dataset.

To gain insights into the generated dataset, Figure 3 shows the dispersion points related to the blocks in the dataset arranged according to their per non-zero execution time and total, package, and DRAM energy consumptions for the different frequencies at which the SpMV blocks were computed. In the plots, we observe that roughly 92% of the total energy corresponds to the package, while the remaining 8% is related to the DRAM component. Also, we detect significant variations in the execution time ranges per non-zero element which decreases with the frequency. A similar effect occurs with all energy consumption metrics. It is also remarkable that the point clouds in the DRAM-related plot follow a less linear trend concerning the time per non-zero element than those displayed in the total and package plots. Similarly, Table 1 displays the execution time and total/package/DRAM energy consumption ranges, means and standard deviations for the blocks in the dataset for the different frequencies. These control values determine the metric ranges that our blockwise SpMV kernel takes to process one unit of work, i.e. a block.

In general, these observations reveal the need for accurate models, as those presented in this work, since not all the metrics for the matrix blocks in the dataset follow a linear trend. Moreover, the distribution obtained in point clouds exposes the need for CNN models that individually estimate the four selected metrics and frequencies.

5.2 Tuning and training the models

The set of CNN models for estimating the selected metrics at the different processor frequencies were implemented using Keras on top of TensorFlow. During the tuning process, we considered the set of hyperparameters listed in Table 2. Specifically, the columns in the table show the different choices for each hyperparameter together with the values estimated by Hyperas for both the execution time and total energy consumption models targeting the CPU frequency 2.4 GHz. Hyperas is a hyperparameter optimization tool for Keras models that operates on top of Hyperopt. An advantage of Hyperas is that, instead of using a grid-based search, it leverages Bayesian search algorithms, such as the tree of Parzen estimators, to partially search the parameter space for relatively optimal parameter settings

Table 2. Hyperparameters considered for the execution time and energy consumption models.

Hyperparameter	Range of values	Execution time models	Energy consumption models
# of filters in blocks of Conv layers	$\{(16, 32), (32, 64)\}$	$(32, 64)$	$(32, 64)$
Filter size in Conv layers	$\{3, 5, 7, 9\} \times 1$	$3 \times 1/9 \times 1$	$5 \times 1/3 \times 1$
Pool size in MaxPool layers	$\{2, 3, 5, 7, 9\} \times 1$	$7 \times 1/5 \times 1$	$7 \times 1/7 \times 1$
# of FC layers	$\{1, 2, 3\}$	2	3
# of neurons in FC layers	$\{10, 100, 1000\}$	1000, 1	100, 1000, 1
Rates in dropout layers	$\mathcal{U}(0, 1)$	6.6×10^{-4}	0.47, 0.59
Optimizer algorithm	$\{SGD, Adam, RMSprop\}$	Adam	Adam
Initial learning rate	$\{10^{-1}, 10^{-2}, 10^{-3}\}$	10^{-3}	10^{-3}
Batch size	$\{32, 64, 128, 256\}$	256	256

(Bergstra et al., 2013). The time taken by Hyperas to perform the hyperparameter search for each model was about a day.

In Table 2, the values in the tuples (f_1, f_2) appearing in the row labeled as "# of filters in blocks of Conv layers" respectively specify the number of filters used in the first (f_1) and second (f_2) Conv layers. Similarly, the values in the last two columns for the hyperparameter labeled as "filter/pool size in Conv/MaxPool layers" stand for the filter sizes in the first and second Conv/MaxPool layers, respectively. On the other hand, the number of values selected for the hyperparameter "# of neurons in FC layers" depends on the value obtained for the parameter "# of FC layers". For instance, the number of neurons for the energy models in the last three FC layers are "100, 1000, 1," respectively. In the same way, the "rates in dropout layers" for these models are 0.47 and 0.59 in each dropout layer. In our experiments, we used an adaptive learning rate that was scaled by 0.1 when the mean squared error (MSE) did not improve any longer. In this sense, the "initial learning rate" refers to the starting value for the learning rate.

Recall that the hyperparameter values for the time model at 2.4 GHz are inherited from those estimating the same metric at lower frequencies. Similarly, the values obtained for the total energy consumption model at 2.4 GHz are leveraged for training all remaining energy models at any other frequency. This way of proceeding is justified given the behavior observed in Figure 3, where, for example, the package energy follows quite a similar trend concerning the total energy. Also, this considerably reduces the number of experiments for calculating the hyperparameters of the models.

Once the values of the hyperparameters are set, the CNN models are ready to be trained. In supervised learning, the entire training dataset is usually divided randomly into the training (in-sample) and validation (out-of-sample) (sub)-sets. The first is used for estimating the model weights (and biases), while the second is only leveraged to guide the training process. Although this practice is well established, the random division of a sample collection may bias the weight estimation, affecting the performance of the models. To this end, an important step prior to the subsequent experimentation is to use a cross-validation scheme to analyze the performance behavior among k data folds. Cross-validation is a technique to assess how the models generalize given an unseen independent dataset (Bishop, 2006). As in Barreda et al. (2020c), we used a 5-fold cross-validation for the time model at 2.4 GHz and observed that the training-validation partitioning does not affect the accuracy of the models, given that there are enough representative data in the considered partitions. Therefore, we selected the first 20% samples of the entire training set for validation. This criterion was adopted for training all remaining models.

The purpose of the training is to minimize a loss function, given the regression-based nature of the models estimating the time and energy per non-zero element. In this work, the selected loss function is the mean squared error (MSE), whose units are ns^2 and nJ^2 for the time and energy-related models, respectively. During the training of the models, this metric is computed for the validation set after each trained batch and leveraged to determine the point where the training shall stop. Concretely, the stopping criteria detains the training process when the MSE does not improve after 15 epochs. With these settings, we observed that the number of epochs needed to reach a MSE plateau and complete the training did not exceed 50 epochs on average for all models. Although this work focuses solely on evaluating the quality of the model estimations, we note that the training process took about 7 hours, being less time-consuming than the hyperparameter search.

5.3 Testing the models

Once the models were trained, the next step was to evaluate them using the testing dataset, which was composed of 34 unseen sparse matrices. This testing step took a few minutes to complete on the compute node, for all matrix blocks in each CNN model. To evaluate the models' accuracy, we use the relative error (RE) as the metric to account for the difference between the predicted and measured values of the modeled metrics. Concretely, the RE for the run time (T) predicted for each block is defined as:

$$RE_{T_b} = \frac{|T^{estimated} - T^{measured}|}{T^{measured}}. \qquad (1)$$

Figure 4 displays the histogram and boxplots of the RE obtained for each of the blocks comprising the matrices in

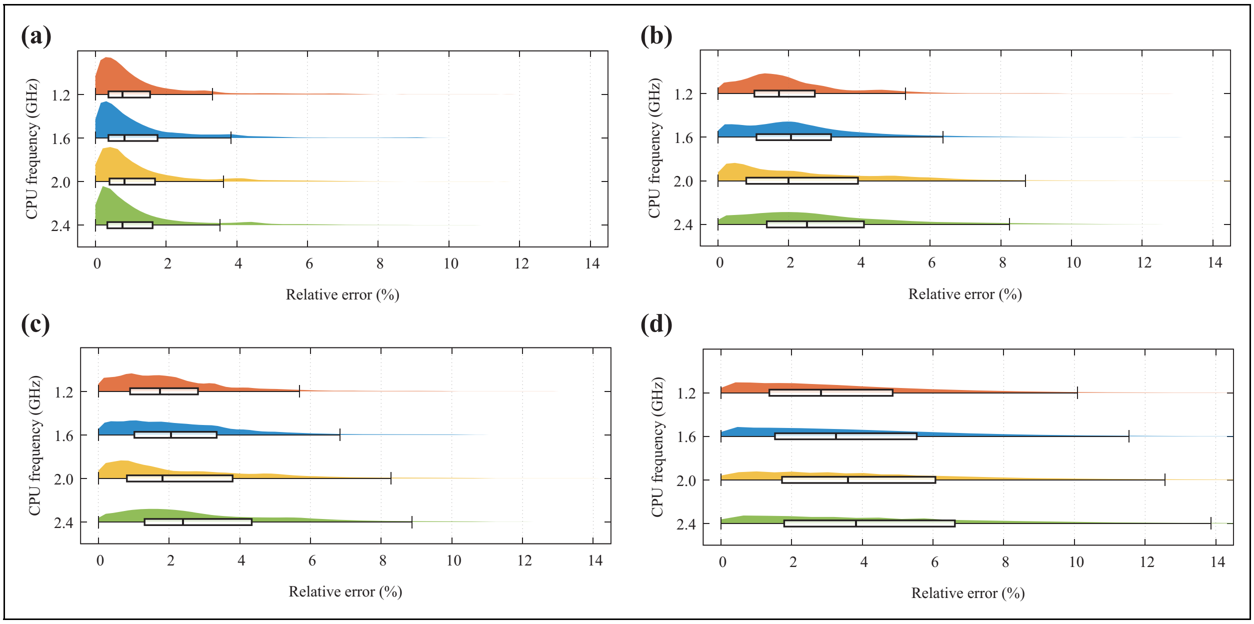

Figure 4. Histogram and boxplots for the estimation relative errors at block level obtained by the CNN models for the selected metrics and frequencies on the testing matrix set. (a) Execution time. (b) Total energy consumption. (c) Package energy consumption. (d) DRAM energy consumption.

the testing dataset. Focusing on the models predicting the execution time, regardless of the frequency, the RE is mostly below 4% and most of the errors are roughly 0.5%. For the total and package energy consumption models, the maximum RE is between 5% and 9%, though most of the samples are, in general, within 2%. In contrast, the RE for the models predicting the DRAM energy consumption have a larger range and are much more dispersed than those in the previous two metrics. Specifically, the maximum RE ranges between 10% and 14% where the samples are slightly settled at 3%. A global observation for the energy-related plots is that the variability of RE notably increases with the frequency. In sum, the RE metrics for the studied models range between 0.5% and 14%, which indicates that the models provide fairly good estimations, even those trained with the inherited hyperparameters.

As a complementary experiment, Figure 5 exposes the RE per test matrix obtained by the trained CNNs for the selected metrics and frequencies. This RE per matrix in the run time is computed as:

$$RE_{T_M} = \frac{\left| \sum_{i=1}^{l} T_i^{estimated} - \sum_{i=1}^{l} T_i^{measured} \right|}{\sum_{i=1}^{l} T_i^{measured}}, \qquad (2)$$

where l is the number of blocks of a given matrix. In this figure, the matrices in the plots are sorted according to the RE delivered by the CNNs modeling the lowest frequency. Concerning the execution time models, the RE is not higher than 10% for all frequencies, while for some matrices the estimation RE is below 0.1%. Similar results can be observed in the models targeting the total and package energy. The CNNs for the DRAM metric, however, deliver slightly less accurate results. This is because the values measured for the DRAM exhibit less predictable behavior, as previously shown in Figure 4d. In global, the results show that the RE is below 1% for roughly half of the tested matrices for all metrics and frequencies. Besides, given that the error is below 10%, the strategy of discarding the last block of the $vpos'$ array when this has less than b entries does not significantly affect the predicted metric.

As the models predicting the execution time provide slightly better estimations than the counterparts for the energy consumption, in a separate study we compared the RE of the energy-related models against a simple analytic energy consumption model that multiplies the average power of the corresponding energy metric by the time predicted by the CNN run time model. The results of this comparison showed differences of less than 0.2% in favor of the analytic model. In general, although the analytic time-based model is able to reach similar accuracy as the CNN model, our claim is that to building such model requires: *i)* to obtain the dataset with both time and energy measurements; *ii)* to compute the average power; and *iii)* to train the CNN run time model. In contrast, the CNN energy models take one step further by directly predicting energy consumption using a dataset that does not need to be labeled with run time measurements.

We have made available our SpMV kernel, datasets, and the Keras modules for the hyperparameter search, training, and testing the CNN models at our GitHub repository (v1.0.0) (Barreda et al., 2020b) and at the Zenodo site (Barreda et al., 2020a).

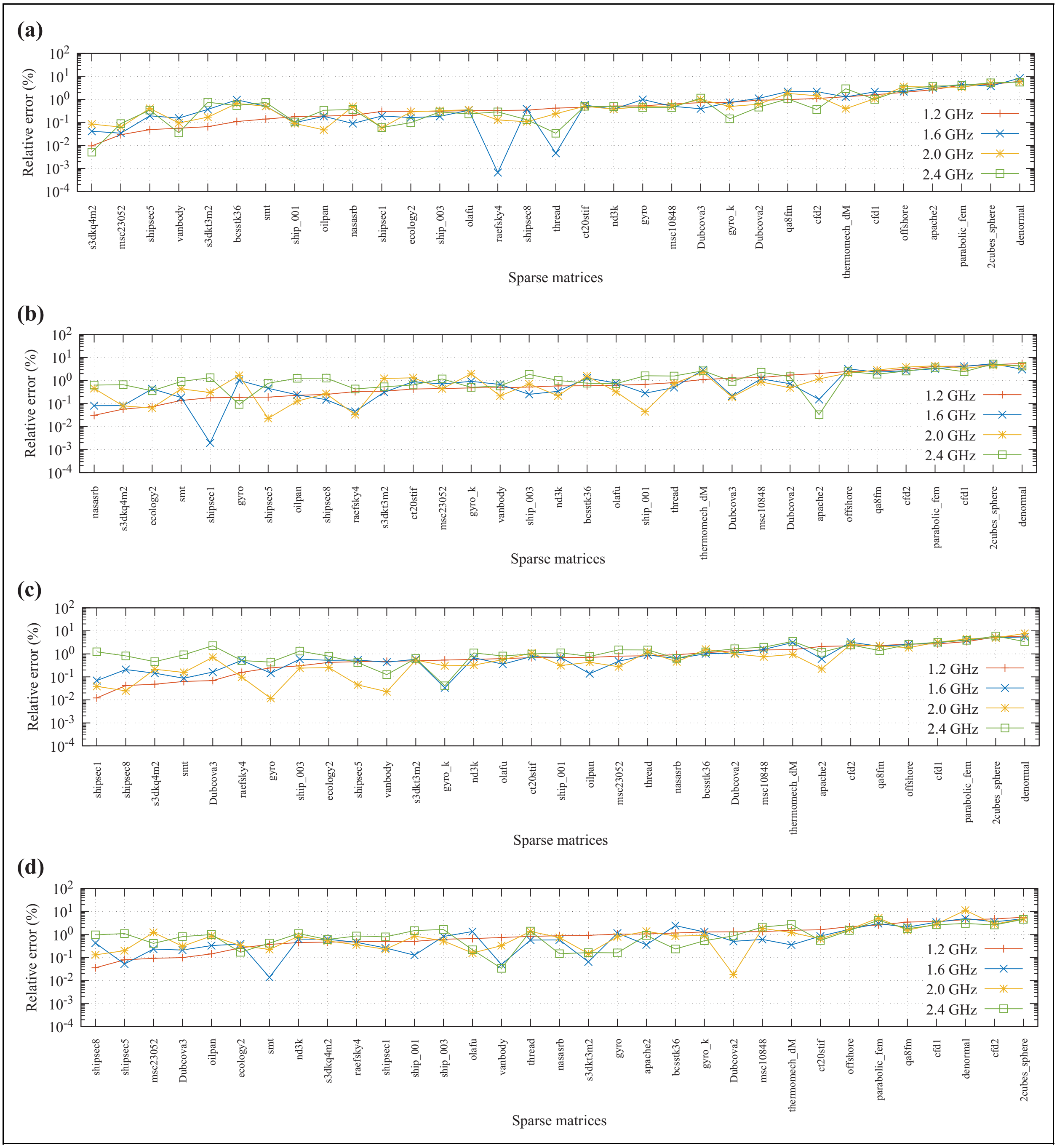

Figure 5. Relative estimation errors at matrix level obtained by the CNN models for the selected metrics and frequencies on the testing matrices. (a) Execution time. (b) Total energy consumption. (c) Package energy consumption. (d) DRAM energy consumption.

6 Use case: the PageRank algorithm

In this section, we derive analytical models for estimating execution time and energy consumption of the PageRank algorithm, a popular method for web information retrieval used by search engines (Bianchini et al., 2005). The reason for selecting this algorithm is twofold: *i)* the core operation in the PageRank algorithm is the SpMV, which allows us to leverage the previously designed CNN-based models; and *ii)* the popularity of the PageRank algorithm together with the time-energy portion that the embedded SpMV consumes, exposes the need for accurate time-energy models.

6.1 The PageRank algorithm

The PageRank algorithm builds upon the Markov chain theory to determine that "a page is relevant if it is linked by other relevant pages." From a practical point of view, PageRank boils down to the classical iterative power

Algorithm 2. Realization of the PageRank algorithm.

```
Require: $A \in \mathbb{R}^{n \times n}, \{p, p'\} \in \mathbb{R}^n, \sigma \in \mathbb{R}^e, \delta, \varepsilon$
for $i = 1, \ldots, n$ do                                    $\triangleright L_0$      $\rightarrow$ flops := 0        memops := $2n$
    $p[i] := \frac{1}{n};\ p'[i] := 0$
end for
$\gamma := 0$
repeat
    $s := 0$
    for $i = 1, 2, \ldots, e$ do                             $\triangleright L_1$      $\rightarrow$ flops := $e$      memops := $2e$
        $s := s + p[\sigma[i]]$
    end for
    for $i = 1, 2, \ldots, n$ do                             $\triangleright SpMV$     $\rightarrow$ flops := $2nnz$   memops := $3nnz$
        for $j = A^T_{vptr[i]}, A^T_{vptr[i]} + 1, \ldots, A^T_{vptr[i+1]} - 1$ do
            $p'[i] := p'[i] + A^T_{vval[j]} \cdot p[A^T_{vpos[j]}]$
        end for
    end for
    for $i = 1, 2, \ldots, n$ do                             $\triangleright L_2$      $\rightarrow$ flops := $n$      memops := $2n$
        $p'[i] := \delta \cdot p'[i] + \frac{(1-\delta)}{n} + \frac{s}{n}$
    end for
    for $i = 1, 2, \ldots, n$ do                             $\triangleright L_3$      $\rightarrow$ flops := $2n$     memops := $2n$
        $\gamma := \gamma + \left|p'[i] - p[i]\right|$
    end for
    for $i = 1, , 2 \ldots, n$ do                            $\triangleright L_4$      $\rightarrow$ flops := 0        memops := $3n$
        $p[i] := p'[i];\ p'[i] := 0$
    end for
until $\gamma < \varepsilon$
```

method, which performs an SpMV involving the sparse matrix reflecting the adjacency graph of the search space (Langville and Meyer, 2006). In the setting of web search, each web page is modeled as a node and the hyperlinks in that web page define directed edges in the model.

Algorithm 2 offers the classic implementation of the PageRank procedure in which $A \in \mathbb{R}^{n \times n}$ is the weighted adjacency (sparse) matrix; $p, p^{'} \in \mathbb{R}^n$ are auxiliary vectors; $\sigma \in \mathbb{R}^e$ is a vector containing the indices of the empty rows of A; δ is the damping factor (usually set to 0.85); and ε is the stopping threshold, which determines the number of iterations and the algorithm precision.

6.2 Time and energy models

As shown in Algorithm 2, in addition to the SpMV product (L_{SpMV}), we identify five memory-bounded loops, L_i for $i \in \{0, 1, \ldots, 4\}$, which have been annotated with the corresponding number of flops and memops.[4] To estimate the PageRank run time, we derive an analytical piecewise model with components that individually estimate the execution time of the loops and SpMV kernel, i.e.

$$T_{PR}(A, \kappa) = T_{L_0} + \kappa(T_{SpMV}(A) + \sum_{i=1}^{4} T_{L_i}), \tag{3}$$

where A and κ denote the input matrix and the number of iterations, respectively. To estimate T_{SpMV}, we leverage the previously trained CNN-based time model. On the other hand, the execution time for the rest of the loops can be easily estimated using the roofline model (Ofenbeck et al., 2014), as they all perform sequential memory accesses. This technique utilizes the arithmetic intensity ($I_{L_i} = \frac{\max(flops_{L_i}, 1)}{\max(memops_{L_i} \cdot \phi, 1)}$, where ϕ refers to the data type size in bytes) to infer the upper bound of flops/s at which the loop can be realized based on the peak processor performance (π) and the memory bandwidth (β). Thus, the L_i run time is estimated as $T_{L_i} = \frac{\max(flops_{L_i}, 1)}{\min(\pi, \beta \cdot I_{L_i})}$. Similar to the time model in Equation (3), the energy counterpart is expressed as:

$$\begin{aligned} E_{PR}(A, \kappa) = E_{L_0} + \kappa\left(E_{SpMV}(A) + \sum_{i=1}^{4} E_{L_i}\right) = \bar{P}_{L_0} \cdot T_{L_0} \\ + \kappa\left(E_{SpMV}(A) + \sum_{i=1}^{4} \bar{P}_{L_i} \cdot T_{L_i}\right). \end{aligned} \tag{4}$$

In this case, we empirically assign $\bar{P}_{L_i}$ based on the average power drawn by the processor when executing instructions at the intensity I_{L_i}. To estimate E_{SpMV} *(A)* we use the CNN trained in Section 5 for predicting the total energy consumption.

6.3 Validation

To prepare the experimental setup for the PageRank algorithm, we selected six sparse matrices from the Stanford Network Analysis Platform (SNAP) of the Suite Sparse Matrix Collection, related to social/communication networks and web graphs. We then ran the PageRank

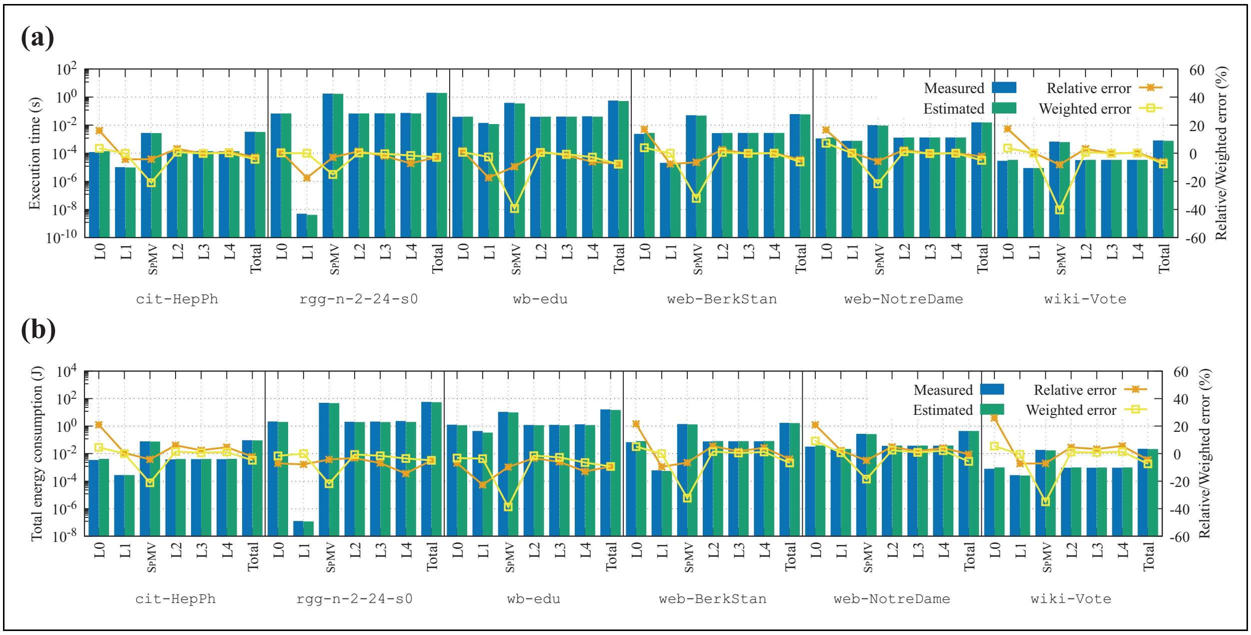

Figure 6. Measured vs estimated PageRank execution time/energy consumption with relative and weighted errors for the testing matrices. (a) Execution time. (b) Total energy consumption.

algorithm with $\delta = 0.85, \varepsilon = 10^{-6}$ for these matrices on the Intel Xeon Haswell core characterized in Section 5, and measured the global and loop-level performance and total energy consumption via Intel RAPL counters. To reduce variability, we computed the average values over 100 executions. In parallel, we fed the related matrix $vpos'$ array to the CNN-based models for predicting the SpMV run time and total energy consumption. The execution time for the rest of the loops was estimated via the roofline model, while the energy consumption was obtained by multiplicating its predicted T_{L_i} by the associated empirical average power $\bar{P}_{L_i}$. For the roofline model we set $\pi = 16\,\text{DPFLOPS/cycle} \cdot 2.4\,\text{GHz} = 38.4\,\text{GFLOPS/s}$[5] as defined by the processor parameters, while we established $\beta = 4.323\,\text{GB/s}$ empirically, according to the memory bandwidth obtained from an independent execution of the STREAM benchmark (McCalpin, 1997–2007). In these experiments, we only targeted the highest CPU frequency, i.e. 2.4 GHz.

To assess the accuracy of our piecewise models, we use both the REs and the weighted relative errors (WRE) at global and loop (including the SpMV kernel) levels. In this specific case, to account for positive and negative execution time relative errors, we drop the absolute value operator from Equation (1), i.e.:

$$RE_{T_{L_i}} = \frac{T_{L_i}^{estimated} - T_{L_i}^{measured}}{T_{L_i}^{measured}}. \tag{5}$$

On the other hand, the WRE for the same metric is defined as follows:

$$WRE_{T_{L_i}} = \frac{T_{L_i}^{estimated} - T_{L_i}^{measured}}{T_{average}}, \tag{6}$$

where the average cost per loop is given as:

$$T_{average} = \frac{T_{SpMV}^{measured} + \sum_{i=0}^{4} T_{L_i}^{measured}}{6}. \tag{7}$$

The expression for WRE_{SpMV} is defined analogously to that in Equation (6) but replacing the term T_{L_i} by T_{SpMV}. With that, the total WRE for the total PageRank algorithm execution time, i.e. T_{PR}, is defined as:

$$WRE_{T_{PR}} = \frac{\left|WRE_{T_{SpMV}}\right| + \sum_{i=0}^{4} \left|WRE_{T_{L_i}}\right|}{6}. \tag{8}$$

In this regard, WRE offers more an insightful view than RE, as it highlights critical REs while smoothens others with little impact on the overall execution time. For instance, when $T_{L_i}^{estimated} > T_{average}$, $WRE_{T_{L_i}}$ carries more weight than $RE_{T_{L_i}}$. Conversely, $WRE_{T_{L_i}}$ results in less weight than $RE_{T_{L_i}}$ when $T_{L_i}^{measured} < T_{average}$. Alternatively, the corresponding expressions for calculating the energy consumption WREs are defined as in Equations (6) and (8), but using E_{L_i} instead.

Figure 6 shows the actual (i.e., measured) vs estimated performance/energy consumption per loop iteration (left Y-axis, in log scale) along with their relative and weighted errors at both loop and global levels (right Y-axis). Note the scale $[-60\%, +60\%]$ used in the right Y-axis for the (W)RE, where negative errors correspond to underestimations while positive errors reflect overestimations. A first inspection of the plots reveals that the REs for both execution time and energy consumption are, in general, not larger than 10%, indicating that both the CNN and roofline-based models deliver fairly good estimations at both loop and global levels. Focusing on WREs, however, we detect relevant underestimations for both T_{SpMV} and E_{SpMV}, of about -40%. This is because the SpMV kernel consumes a considerable portion of the overall time and energy of the PageRank algorithm, concretely from roughly 63% to 85% for the smallest and largest matrices, respectively. The WRE for the rest of the loops is, in contrast, smoothened. In the light of the results, we can consider that the analytical performance and energy consumption models, combining the CNN-based ones, provide fairly accurate estimations for the PageRank algorithm. We also note that our CNN-based models can be useful to estimate any other algorithm internally realizing the SpMV in its core.

7 Conclusions

In this paper, we have presented a series of CNNs to predict the execution time and the energy consumption of the SpMV kernel, a memory-bounded operation that is fundamental in a broad range of scientific and engineering problems. We demonstrate that the CNN-based models can capture the intricate patterns and features of the sparse matrix (stored in CSR format), which dictate the irregular accesses to the dense input vector. Specifically, our models target the execution time and the total, package, and DRAM energy, measured via Intel RAPL counters at four different processor frequencies. In addition, we employ the PageRank algorithm as a use case to build high-level estimators for run time and energy consumption that combine two techniques: the well-known roofline-model and our SpMV CNN-based models.

During the experimental evaluation, we calculated the hyperparameters for the models targeting the execution time and total energy consumption metrics at the highest frequency. The rest of the models predicting run time and total, package, and DRAM energy at lower frequencies inherited the hyperparameters. With them, we trained the networks using the sparse matrices blocks from the SuiteSparse matrix collection labeled with the aforementioned metrics values corresponding to the blockwise realization of the SpMV on an Intel Xeon Haswell core. In general, the RE metrics for the studied models range between 0.5% and 14%, which indicates that the models provide fairly good estimations, even those trained with the inherited hyperparameters. We observed that the models targeting higher frequencies for all metrics, and especially those modeling the DRAM energy component, delivered slightly higher RE values compared with the rest. The study performed for the testing matrices revealed that RE was always below 1% for roughly half of the tested matrices and all metrics and frequencies. The time-energy models designed for the PageRank algorithm showed that our CNN models can be integrated into more complex ones, delivering fair results, which can then be leveraged to perform off-line estimations of the desired metric.

All in all, the main benefit of our time-energy CNN-based models is that, given trained models for a variety of processor architectures, selecting the best option (processor and configuration, e.g. operation frequency) does not require direct access to the target platforms. However, we also acknowledge some weaknesses: *i)* the inference costs (for both runtime and energy consumption) are considerably higher than realizing the SpMV on the target architecture; and *ii)* the proposed blockwise SpMV algorithm is slightly less efficient than the current state-of-the-art SpMV implementations. Nevertheless, we believe that estimating the execution cost of irregular operations, such as the SpMV, paves the way toward applying similar ML-driven techniques to modeling the cost of more complex numerical kernels when no access to the target platform is provided.

As future work, we plan to extend the CNNs to estimate the execution time and energy consumption of the parallel implementation of the SpMV operation. An additional goal is to leverage this methodology to model both run time and energy consumption of more complex linear algebra operations targeting a wider range of platforms. To avoid the hyperparameter optimization costs, we will migrate pre-trained models to estimate the performance/energy consumption of other computing architectures.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by project TIN2017-82972-R from the *MINECO*, Spain. Manuel F Dolz was also supported by the Plan GenT project CDEIGENT/2018/014 from the *Generalitat Valenciana*, Spain. Maria Barreda was also supported by the POSDOC-A/2017/11 project from the *Universitat Jaume I*.

ORCID iD

Manuel F Dolz https://orcid.org/0000-0001-9466-3398

Notes

1. The row-density is number of non-zeros of per row (*nzr*) relative to the row size (*n*), i.e. $\frac{nzr}{n} \in [0, 1]$.
2. The compressed row/column formats are probably the most popular arrangements for storing general sparse matrices, as they make absolutely no assumptions about the sparsity structure of the matrix and do not store any unnecessary elements. These formats are well supported in many numerical computing environments, such as Matlab, Octave or Julia.
3. We assume that the "model" term comprises the combination of CNN architecture and its corresponding numerical values for weights and bias.
4. A memop accounts for read and write memory operations. We do not distinguish between the memop type.

5. The DP FLOPS unit refers to double-precision floating-point operations.

References

Abadi M, Agarwal A, Barham P, et al. (2015) TensorFlow: large-scale machine learning on heterogeneous systems. Available at: https://www.tensorflow.org/. Software available from tensorflow.org (accessed July 2020).

Barreda M, Dolz MF and Castaño MA (2020a) Block-wise sparse matrix-vector product dataset and convolutional neural nets for estimating the run time and energy consumption of the sparse matrix-vector product. Available at: https://doi.org/10.5281/zenodo.3956057 (accessed July 2020).

Barreda M, Dolz MF and Castaño MA (2020b) Convolutional neural nets for estimating the run time and energy consumption of the sparse matrix-vector product-spmv-cnn. Available at: https://github.com/hpca-uji/SpMV-CNN-Model/releases/tag/v1.0.0.

Barreda M, Dolz MF, Castaño MA, et al. (2020c) Performance modeling of the sparse matrix-vector product via convolutional neural networks. *The Journal of Supercomputing*. Epub ahead of print 04 February 2020. DOI: 10.1007/s11227-020-03186-1.

Benatia A, Ji W, Wang Y, et al. (2016) Machine learning approach for the predicting performance of SpMV on GPU. In: *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, Wuhan, China, 13–16 December 2016, pp. 894–901. DOI:10.1109/ICPADS.2016.0120.

Bergstra J, Yamins D and Cox DD (2013) Hyperopt: a Python library for optimizing the hyperparameters of machine learning algorithms. In: *Proceedings of the 12th Python in science conference*, volume 13, Austin, Texas, USA, 24–29 June 2013, p. 20. Citeseer.

Bianchini M, Gori M and Scarselli F (2005) Inside PageRank. *ACM Transactions on Internet Technology* 5(1): 92–128.

Bishop CM (2006) *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag. ISBN 0387310738.

Boureau Y, Ponce J and Lecun Y (2010) A theoretical analysis of feature pooling in visual recognition. In: *ICML 2010—Proceedings, 27th International Conference on Machine Learning, ICML*, Haifa, Israel, 21–24 June 2010. ISBN 9781605589077, pp. 111–118.

Chollet F. (2015) Keras. Available at: https://github.com/fchollet/keras (accessed July 2020).

Cui H, Hirasawa S, Kobayashi H, et al. (2018) A machine learning-based approach for selecting SpMV kernels and matrix storage formats. *IEICE Transactions on Information and Systems* E101.D(9): 2307–2314.

Davis TA and Hu Y (2011) The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software* 38(1): 1–25.

Eijkhout V and Pozo R (1994) Data structures and algorithms for distributed sparse matrix operations. Technical report. University of Tennessee, Knoxville, TN, USA.

Elafrou A, Goumas G and Koziris N (2017) Performance analysis and optimization of sparse matrix-vector multiplication on modern multi- and many-core processors. In: *2017 46th International Conference on Parallel Processing (ICPP)*, Bristol, UK, 14–17 September 2017, pp. 292–301. DOI: 10.1109/ICPP.2017.38.

Endrei M, Jin C, Dinh M, et al. (2019) Statistical and machine learning models for optimizing energy in parallel applications. *The International Journal of High Performance Computing Applications* 33: 1079–1097.

Gkountouvas T, Karakasis V, Kourtis K, et al. (2013) Improving the performance of the symmetric sparse matrix-vector multiplication in multicore. In: *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, Boston, MA, USA, 20–24 May 2013, pp. 273–283. DOI:10.1109/IPDPS.2013.43.

Glorot X, Bordes A and Bengio Y (2011) Deep sparse rectifier neural networks. In: Gordon G, Dunson D and Dudk M (eds) *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, Proceedings of Machine Learning Research*, volume 15. Fort Lauderdale, FL, USA: PMLR, pp. 315–323. URL http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf (accessed July 2020).

Götz M and Anzt H (2018) Machine learning-aided numerical linear algebra: convolutional neural networks for the efficient preconditioner generation. In: *Proceedings of ScalA'18: Ninth Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, WS at Supercomputing*. Dallas, Texas, USA, 12 November 2018, DOI: 10.13140/RG.2.2.30244.53122.

Gu J, Wang Z, Kuen J, et al. (2018) Recent advances in convolutional neural networks. *Pattern Recognition* 77(C): 354–377.

Langville AN and Meyer CD (2006) *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton, NJ: Princeton University Press. ISBN 0691122024.

LeCun Y, Bengio Y and Hinton G (2015) Deep learning. *Nature* 521: 436–444.

Lee C, Gallagher PW and Tu Z (2015) Generalizing pooling functions in convolutional neural networks: mixed, gated, and tree. *CoRR* abs/1509.08985. URL http://arxiv.org/abs/1509.08985 (accessed July 2020).

Li K, Yang W and Li K (2015) Performance analysis and optimization for SpMV on GPU using probabilistic modeling. *IEEE Transactions on Parallel and Distributed Systems* 26(1): 196–205.

Llopis P, Dolz MF, Blas JG, et al. (2016) Analyzing the energy consumption of the storage data path. *The Journal of Supercomputing* 72(11): 4089–4106.

Malossi ACI, Ineichen Y, Bekas C, et al. (2014) Performance and energy-aware characterization of the sparse matrix-vector multiplication on multithreaded architectures. In: *2014 43rd International Conference on Parallel Processing Workshops*, Minneapolis, MN, USA, 9–12 September 2014, pp. 139–148. DOI:10.1109/ICPPW.2014.30.

McCalpin JD (1991–2007) Stream: sustainable memory bandwidth in high performance computers. Technical report. Charlottesville, Virginia: University of Virginia. Available at: http://www.cs.virginia.edu/stream/. A continually updated technical report. http://www.cs.virginia.edu/stream/ (accessed July 2020).

Nisa I, Siegel C, Rajam AS, et al. (2018) Effective machine learning based format selection and performance modeling for SpMV on GPUs. *EasyChair Preprint* no. 388. DOI: 10.29007/lnnt.

Ofenbeck G, Steinmann R, Caparros V, et al. (2014) Applying the roofline model. In: *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Monterey, CA, USA, 23–25 March 2014, pp. 76–85. DOI: 10.1109/ISPASS.2014.6844463.

Pumperla M (2017) Hyperas: a very simple convenience wrapper around hyperopt for fast prototyping with Keras model. Available at: http://maxpumperla.com/hyperas/ (accessed July 2020).

Schmidhuber J (2015) Deep learning in neural networks: an overview. *Neural Networks* 61: 85–117.

Song S, Su C, Rountree B, et al. (2013) A simplified and accurate model of power-performance efficiency on emergent GPU architectures. In: *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, Boston, MA, USA, 20–24 May 2013, pp. 673–686. DOI: 10.1109/IPDPS.2013.73.

Tiwari A, Laurenzano MA, Carrington L, et al. (2012) Modeling power and energy usage of HPC kernels. In: *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, Shanghai, China, 21–25 May 2012, pp. 990–998. DOI: 10.1109/IPDPSW.2012.121.

Zhao Y, Li J, Liao C, et al. (2018) Bridging the gap between deep learning and sparse matrix format selection. *SIGPLAN Notice* 53(1): 94–108.

Author biographies

Maria Barreda is a postdoctoral researcher at Universitat Jaume I. She obtained her Ph.D. in Advanced Computer Systems in the Universitat Jaume I, in 2017. During her research career, she has worked in energy efficiency techniques applied to high-performance computing applications, as well as in the analysis and optimization of algorithms and sparse linear algebra solvers in different types of processors. Maria has participated in several projects funded by the Spanish government, although her participation in the EU H2020—FETHPC—INTERTWinE project stands out. In this project, her main research line has been the interoperability between programming models. Moreover, she did internships at the Technische Universität of Braunschweig and INRIA, Paris. Nowadays, her research interests are focused on reproducibility techniques and deep learning in HPC.

Manuel F Dolz received his Ph.D. in Advanced Computer Systems from the Universitat Jaume I (Spain) in 2014. Currently, he is a distinguished researcher at the same university. During his career, he worked as a pre and postdoctoral researcher at the University of Hamburg and University of Carlos III Madrid (UC3M) for the EU projects Exa2Green and RePhrase, respectively. Manuel has also participated in other R+D+i projects at national and regional levels. His main research interests are parallel programming environments, energy efficiency, and deep learning for the high-performance parallel computing domain. He collaborates with national universities (Technical University of Valencia and UC3M) and international institutions (Deutsches Klimarechenzentrum-University of Hamburg, Germany, and École Normale Supériere, Lyon, France). Manuel has participated in different international conferences and workshops program committees and acted as a reviewer in international conferences and journals indexed in JCR. In total, he has published more than 75 articles in conferences and national and international journals, 24 of them indexed in JCR.

M Asunción Castaño received her B.S. degree from the Universidad Politécnica de Valencia (Spain) in 1991 and her Ph.D. in Computer Science in 1998 from the same University. She has participated in different EU, national and regional projects focused on both research and education. M^a Asunción also belongs to the Steering Committee and the Scientific Committee of several conferences. She focuses her research on deep learning, energy saving on HPC platforms, and educational innovation.