Energy-Efficient and High-Throughput CNN Inference on Embedded CPUs-GPUs MPSoCs

Erqian Tang(✉), Svetlana Minakova, and Todor Stefanov

LIACS, Leiden University, Leiden, The Netherlands
{e.tang,s.minakova,t.p.stefanov}@liacs.leidenuniv.nl

Abstract. Nowadays, many application scenarios, such as mobile phones, drones, mobile robots, require Convolutional Neural Networks (CNNs) inference on embedded CPUs-GPUs MPSoCs. CNN model inference is usually computation intensive while the embedded CPUs-GPUs MPSoCs are usually energy consumption constrained. Therefore, how to achieve computationally-intensive CNN inference in an energy-efficient and high-throughput way is an important issue. However, existing Deep Learning (DL) frameworks only pay attention to achieving high-throughput inference when deploying CNN models on CPU or GPU processors without specifically considering the energy consumption.

In this paper, we propose a novel methodology which features design-time optimization techniques in order to achieve energy efficiency and high throughput when deploying CNN models on embedded CPUs-GPUs MPSoCs. Our methodology finds Pareto-optimal mappings of a CNN model onto a CPUs-GPUs MPSoC with voltage and frequency scaling (VFS) configurations with the help of a two-objective Genetic Algorithm (GA) which optimizes the system throughput and energy consumption simultaneously. Moreover, we propose two analytical models, that are used as fitness functions in the two-objective GA to evaluate very fast the system throughput and energy consumption of CNNs mapped onto embedded CPUs-GPUs MPSoCs. Also, we confirm the high accuracy of these two analytical models by experimental evidence. Finally, our experimental results show that our novel methodology is able to achieve both energy efficiency and high throughput when deploying CNN models on embedded CPUs-GPUs MPSoCs, in comparison with TensorRT which is the best-known CNN deployment optimizer designed for NVIDIA embedded MPSoCs.

Keywords: Convolutional Neural Networks · SDF · Pareto-optimal mapping · High-throughput · Energy-efficient · MPSoCs · TensorRT

1 Introduction

Convolutional Neural Networks (CNNs) are biologically inspired graph computational models, characterized by high degree of available parallelism. Due to their

A. Orailoglu et al. (Eds.): SAMOS 2021, LNCS 13227, pp. 127–143, 2022.
https://doi.org/10.1007/978-3-031-04580-6_9

ability to handle large, unstructured data, CNNs are widely used to perform various tasks in areas such as computer vision and natural language processing [1]. The CNNs execution typically includes two phases: training and inference [1]. At the training phase the optimal CNN parameters are established. At the inference phase, a trained CNN is applied to the actual data and performs the task for which the CNN is designed. Due to the high complexity of state-of-the-art CNNs, their training and inference phases are usually performed by high-performance platforms, and provided as cloud services. However, some applications, e.g. [2–4], require high-throughput execution of the CNNs inference, which cannot be provided as a cloud service. These applications are typically deployed on embedded devices.

Many modern embedded devices are based on multi-processor systems-on-chip (MPSoCs) [5]: complex integrated circuits, that consist of processing elements with specific functionalities. Due to their specific design, MPSoCs offer energy-efficient and high-throughput solutions for applications running on embedded devices. In addition to hosting various processing elements, capable of running the CNN inference, such as central processing units (CPUs), embedded graphics processing units (embedded GPUs), and field-programmable gate arrays (FPGAs), MPSoCs integrate many other components, such as communication network components and video accelerators, that allow to deploy the entire embedded application on a single chip. Therefore, MPSoCs seem to be a promising solution for the deployment of the CNN inference phase on embedded devices.

Embedded CPUs-GPUs MPSoCs are usually energy consumption constrained while the CNN model inference is computation intensive. For example, in many application scenarios requiring CNN inference on embedded MPSoCs, such as mobile phones, drones, mobile robots, the battery capacity is usually very limited. So, how to achieve computationally-intensive CNN inference in an energy-efficient way on embedded CPUs-GPUs MPSoCs is an important issue.

However, existing Deep Learning (DL) frameworks [6–16], that enable execution of the CNN inference on embedded CPUs-GPUs MPSoCs, only pay attention to achieving high-throughput inference when deploying CNN models on CPU or GPU processors without specifically considering the energy consumption, which can be influenced by the utilized number of processors and by the possibility for CPUs-GPUs voltage and frequency scaling (VFS). These frameworks rely on the operating system to determine the utilized number of processors and the CPUs-GPUs operating frequency at run-time and do not support design-time optimizations for energy-efficient deployment of the CNN inference.

Therefore, in this paper, we extend the methodology in [16], with design-time optimization techniques in order to achieve energy efficiency and high throughput when deploying CNN models on embedded CPUs-GPUs MPSoCs. We propose to extend [16] because it exploits explicitly both task- and data-level parallelism, available in a CNN, thereby achieving higher throughput compared to the other existing DL frameworks [6–15]. We exploit this higher throughput in combination with different CPUs and GPUs utilization and VFS configuration possibilities to

reduce the energy consumption and to optimize the CNN inference on an MPSoC at design-time. The goal of our optimization is to find an MPSoC configuration which achieves the same or higher throughput with less energy consumption compared to existing DL frameworks.

Paper Contributions
In this paper, we extend the methodology in [16], which consists of three main steps, introduced in Sect. 3.1. In Step 1, a CNN model is automatically converted to a functionally equivalent Syndchronous Dataflow (SDF) model. In Step 2, an efficient mapping of the SDF model onto a CPUs-GPUs MPSoC is obtained using a single-objective genetic algorithm (GA) to achieve high throughput by utilizing the hardware resources as much as possible. Our main novel contributions are related to Step 2 and include: 1) We propose to use a two-objective GA in order to optimize for system throughput and energy consumption simultaneously. To enable such two-objective GA-based optimization, we propose novel and very accurate analytical models and use them as fitness functions in the GA to evaluate very fast the system throughput and energy consumption of CNNs mapped onto embedded CPUs-GPUs MPSoCs; 2) We confirm the high accuracy of our aforementioned analytical models by comparing the system throughput and energy consumption numbers provided by our models with measured numbers obtained by deploying real-world CNNs on the Nvidia Jetson-TX2 embedded platform; 3) We use the extended methodology and models, mentioned above, to find pareto-optimal mappings of real-world CNNs onto the Nvidia Jetson-TX2 MPSoCs platform. The obtained results, in terms of system throughput and energy consumption, are compared with results obtained by the best-known DL framework for Jetson MPSoCs called TensorRT [15]. This comparison shows that our extended methodology can achieve CNN inference on embedded CPUs-GPUs MPSoCs with the same or higher throughput but with less energy consumption.

The remainder of the paper is organized as follows: Sect. 2 gives an overview of the related work. Section 3 introduces the background material needed for understanding the contributions of this paper. Section 4 presents our proposed extension of the methodology in [16], briefly introduced in Sect. 3.1, including our two novel analytical models. Section 5 shows the experimental results and Sect. 6 ends the paper with conclusions.

2 Related Work

Among the existing DL frameworks [6–15], NVidia TensorRT [15] is the best-known CNN deployment optimizer designed for embedded MPSoCs such as Nvidia Jetson TX2. This optimizer is built on top of CUDA and includes several optimizations techniques to deliver high throughputs and low latencies for deep neural network applications. TensorRT tries to minimize the memory footprint of a CNN by reusing memory and applying fusion operations. Also, it exploits data-level parallelism, available in a CNN, for efficient utilization of embedded GPUs. However, TensorRT relies on layer-by-layer (sequential) execution of CNN layers

and only one CPU processor is utilized for launching GPU engines and sending data. In this way, the available CPUs-GPUs MPSoC hardware resources are not utilized in the most efficient way in terms of high-throughput. Moreover, the focus of TensorRT is only to improve the system throughput, so optimizing the energy consumption is not considered. The CPUs and GPUs processors do not operate at proper frequency configurations, which is not a good solution for some energy constrained DL applications, such as drones or other light battery mobile robots. In contrast, our extended methodology exploits both data-level parallelism within the same layer and task-level parallelism among different layers of a CNN. At the same time, our methodology also considers different number of processors to be utilized and different CPUs-GPUs VFS to be applied. Therefore, compared to TensorRT, our methodology can achieve same or better inference system throughput, and lower energy consumption at the same time. Moreover, our extended methodology is implemented on top of TensorRT, thereby inheriting some benefits of TensorRT as well, such as minimizing the memory footprint and applying fusion operations.

In [16], a novel methodology for execution of the CNN inference on embedded CPUs-GPUs MPSoCs is proposed. It takes full advantage of all CPU and GPU resources, available in an MPSoC, and ensures high-throughput CNN inference execution on CPUs-GPUs MPSoCs by efficiently exploiting task-level (pipeline) parallelism, available among CNN layers, together with data-level parallelism, available within CNN layers. [16] achieves higher CNN inference throughput on embedded CPUs-GPUs MPSoCs compared to the aforementioned NVidia TensorRT [15] deployment optimizer. However, utilizing all possible CPU and GPU resources increases the energy consumption, which may fail to meet the energy consumption budget of some battery constrained applications. In contrast, in this paper, we extend [16] in order to enable fast and accurate multi-objective design space exploration to find more efficient utilization of CPU and GPU resources at proper VFS configurations. Therefore, we reduce the energy consumption while still achieving high CNN inference throughput.

3 Background

In this section, we briefly introduce the methodology in [16] for execution of the CNN inference on embedded CPUs-GPUs MPSoCs as well as we describe the specific features of the embedded CPUs-GPUs MPSoCs, we consider in this paper, and the Synchronous Dataflow (SDF) model [17]. All these are essential for understanding our paper contributions.

3.1 CNN Inference on Embedded CPUs-GPUs MPSoCs

[16] proposes a novel methodology to deploy a CNN model on embedded CPUs-GPUs MPSoCs. This methodology consists of three main steps. An overview of this methodology is shown in Fig. 1. In Step 1, a CNN model is converted into a functionally equivalent Synchronous Dataflow (SDF) model. Unlike the

CNN model, the SDF model explicitly specifies task- and data-level parallelism, available in a CNN, as well as it explicitly specifies the tasks communication and synchronization mechanisms, suitable for efficient mapping and execution of a CNN on an embedded MPSoC. Thus, a conversion of a CNN model into a SDF model is necessary for efficient mapping and execution of a CNN on an embedded CPUs-GPUs MPSoC. In Step 2, a Genetic Algorithm (GA) is utilized to find an efficient mapping of the SDF model, obtained on Step 1, on an embedded CPUs-GPUs MPSoC. The mapping, obtained by the GA, describes the distribution of the CNN inference computational workload on an embedded MPSoC, that exploits efficiently both task-level and data-level parallelism, available in the CNN. In Step 3, the mapping obtained in Step 2 is utilized to convert a CNN model into a final platform-aware executable Cyclo-Static Dataflow (CSDF) application model [18]. The CSDF model, obtained in Step 3, describes the CNN inference as an executable application, efficiently distributed over embedded MPSoC processors and exploiting the right amount of task- and data-level parallelism, which matches the computational capacity of an embedded MPSoC. Thus, this methodology takes full advantage of all CPU and GPU resources, available in an MPSoC, and enables high-throughput execution of the CNN inference on embedded CPUs-GPUs MPSoCs.

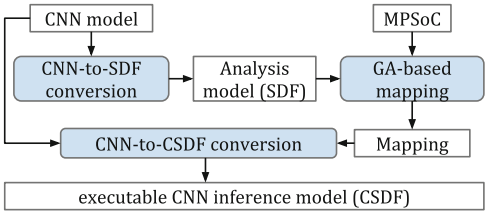

Fig. 1. An overview of the methodology in [16]

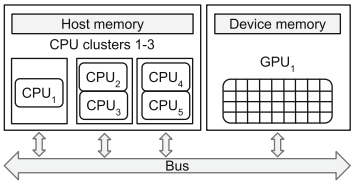

Fig. 2. Embedded MPSoC

3.2 Embedded CPUs-GPUs MPSoCs

We define an embedded MPSoC as a tuple $MPSoC(cpu, gpu)$, where $cpu = \{cpu_1, cpu_2, ..., cpu_n\}$ is a set of all CPU cores, available in the MPSoC; $gpu = \{gpu_1, gpu_2, ..., gpu_m\}$ is a set of all GPU devices, available in the MPSoC, and typically $m \leq n$. An example of an embedded CPUs-GPUs MPSoC with $n = 5$ CPU cores and $m = 1$ GPU device is shown in Fig. 2. CPU cores are usually divided into several clusters. The CPU cores in a cluster can operate at one of the frequencies from the set $f_{cpu} = \{f_{c1}, f_{c2}, ..., f_{cp}\}$. Each GPU can also operate at one of the frequencies from the set $f_{gpu} = \{f_{g1}, f_{g2}, ..., f_{gq}\}$.

3.3 Synchronous Dataflow (SDF) Model

The SDF model [17] is a well-known dataflow model of computation, widely used in the embedded systems community for efficient mapping of applications

on embedded devices, including embedded CPUs-GPUs MPSoCs. An application, modeled as a SDF, is a directed graph $G(A, C)$, which consists of a set of nodes A, also called actors, communicating through a set of FIFO channels C. An example of a SDF model with $|A|$=23 actors and $|C|$=24 FIFO channels is given in Fig. 3. Every actor $a_i \in A$ is a task, which performs certain application functionality, represented as a function F_i. An example of SDF actor a_3 is shown in Fig. 3. Actor a_3 performs function $F_3 = \{ReLU\}$. Every FIFO channel $c_{ij} \in C$ represents data dependency and transfers data in tokens between actors a_i and a_j. c_{ij} has data production rate U_{ij} and data consumption rate V_{ij}. U_{ij} specifies the production of data tokens into channel c_{ij} by actor a_i. V_{ij} specifies the consumption of data tokens from channel c_{ij} by actor a_j. An example of a communication FIFO channel c_{36} is shown in Fig. 3. Channel c_{36} transfers data between actors a_3 and a_6. It has production rate U_{36}=[112640], specifying, that, at each firing, actor a_3 produces 112640 data tokens into channel c_{36} and consumption rate V_{36}=[112640], specifying, that, at each firing, actor a_6 consumes 112640 data tokens from channel c_{36}.

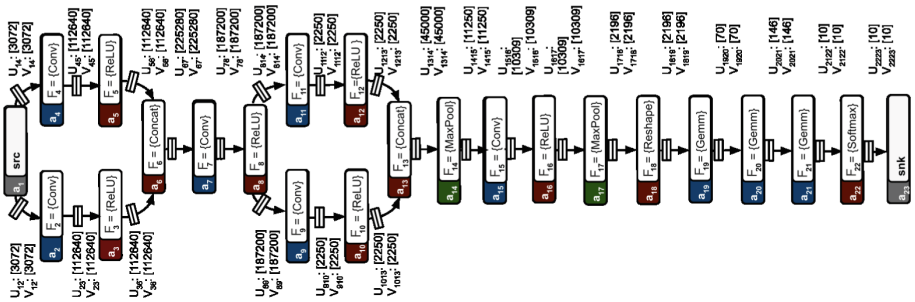

Fig. 3. An example of a SDF model

4 Methodology Extension

In this section, we present our proposed extension of the methodology, introduced in Sect. 3.1, to find an energy-efficient mapping of a SDF model, obtained automatically from a CNN model, onto an embedded CPUs-GPUs $MPSoC(cpu, gpu)$, defined in Sect. 3.2, with a proper configuration (proper utilization of the processors and VFS). In order to achieve this, first, we give our definition of a mapping of a SDF model onto an MPSoC with certain VFS configuration - see Sect. 4.1. Then, we use a two-objective GA in order to optimize the MPSoC system throughput and energy consumption simultaneously - see Sect. 4.2. To enable such two-objective GA-based optimization, we propose novel and very accurate analytical models, see Sect. 4.3, and use them as fitness functions in the GA to evaluate very fast the system throughput and energy consumption of CNNs mapped onto embedded CPUs-GPUs MPSoCs.

4.1 Mapping with VFS Configuration

In our extended methodology, the CNN inference tasks, explicitly specified as SDF actors, are executed on embedded CPU cores, that are able to efficiently handle the task-level parallelism among the different tasks. To efficiently utilize the data-level parallelism, available within the tasks, some of the CPU cores offload computations on the embedded GPUs. Since the number of embedded GPU devices is limited, it may occur, that the efficient exploitation of task-level parallelism, by embedded CPUs, is disrupted due to CPUs competition for the limited embedded GPU devices. To avoid such disruption, for every embedded GPU $gpu_j \in gpu$, we allocate a single CPU core $cpu_i \in cpu$, which offloads computations on gpu_j.

Based on the discussion above, we define a mapping of SDF model $G(A, C)$ onto $MPSoC(cpu, gpu)$ with a VFS configuration, as a partition of actors set A into n subsets, where $n = |cpu|$ is the number of CPU cores, available in the MPSoC. We denote such mapping as ${}^nA = \{{}^nA_1, {}^nA_2, ..., {}^nA_n\}$, where each ${}^nA_i \in {}^nA$ is a subset of actors, mapped on cpu_i, such that $\cap_{i=1}^{n} {}^nA_i = \emptyset$, and $\cup_{i=1}^{n} {}^nA_i = A$. The first $m = |gpu|$ number of CPU cores in mapping ${}^nA$ offload computations on the corresponding embedded GPUs, i.e., the computations within every actor $a_k \in {}^nA_j, j \in [1, m]$ are performed on gpu_j, and the computations within every actor $a_k \in {}^nA_i, i \in [m+1, n]$ are performed on cpu_i. Each cpu_i operates at a frequency $f_{cp} \in f_{cpu}$, and CPUs of the same cluster operate at the same frequency. Each gpu_j operates at a frequency $f_{gq} \in f_{gpu}$.

An example of a mapping with a VFS configuration, ${}^5A = \{{}^5A_1, {}^5A_2, {}^5A_3, {}^5A_4, {}^5A_5\}$ of the SDF model $G(A, C)$, shown in Fig. 3 and explained in Sect. 3.3, on the embedded MPSoC, shown in Fig. 2 and explained in Sect. 3.2, is given in Table 1. In this example, we consider that $f_{cpu} = \{f_{c1}, f_{c2}, f_{c3}, f_{c4}\}$ and $f_{gpu} = \{f_{g1}, f_{g2}, f_{g3}, f_{g4}, f_{g5}\}$. Every Column in Table 1 corresponds to a subset ${}^5A_i, i \in [1, 5]$. For example, Column 1 in Table 1 corresponds to subset ${}^5A_1 = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$. The actors within subset ${}^5A_1$ are mapped on cpu_1, which offloads computations on gpu_1. cpu_1 and gpu_1 operate at frequencies f_{c1} and f_{g2}, respectively. Column 2 in Table 1 describes subset ${}^5A_2 = \{a_8, a_9, a_{10}, a_{13}\}$. Every actor $a_i \in {}^5A_2$ is mapped on cpu_2, and cpu_2 operates at frequency f_{c3}. Since the MPSoC does not have gpu_2, all computations within actors in ${}^5A_2$ are performed only on cpu_2. Since cpu_2 and cpu_3 belong to the same cluster, as shown in Fig. 2, cpu_3 also operates at frequency f_{c3}. Similarly, cpu_4 and cpu_5 operate at frequency f_{c4}.

Table 1. Example of a Mapping with a VFS configuration

$cpu_1@f_{c1}/gpu_1@f_{g2}$	$cpu_2@f_{c3}$	$cpu_3@f_{c3}$	$cpu_4@f_{c4}$	$cpu_5@f_{c4}$
$a_1, a_2, a_3, a_4, a_5, a_6, a_7$	a_8, a_9, a_{10}, a_{13}	a_{11}, a_{12}	$a_{14}, a_{15}, a_{16}, a_{17}, a_{18}, a_{21}, a_{22}, a_{23}$	a_{19}, a_{20}

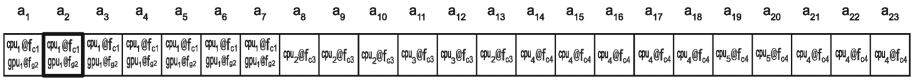

Fig. 4. Mapping chromosome example

4.2 Two-objective GA Optimization

With the aforementioned mapping in Sect. 4.1, we associate two system characteristics, (1) the system throughput: the amount of data processed per unit of time, for example measured in images per second (img/s); (2) the system energy consumption: the total energy needed to process a unit of data, for example measured in joules per image (J/img). We assume a mapping to be efficient if the system throughput is maximized and the system energy consumption is minimized. As these two objectives are conflicting, i.e., the increase of the throughput will cause increase of the energy consumption, we note, that obtaining such an efficient mapping of an SDF graph onto a CPUs-GPUs MPSoC with a VFS configuration is not possible. Thus, we have to perform a complex Design Space Exploration (DSE) in order to find a set of Pareto-optimal mappings [19] that we will consider efficient in our case. In our extended methodology, to solve this problem, we propose to use a two-objective Genetic Algorithm (GA) [20]: a well-known heuristic approach, widely used for finding Pareto-optimal solutions for complex DSE problems. We use a GA with standard two-parent crossover, a single-gene mutation, and standard user-defined GA parameters, such as initial offspring size, number of epochs, mutation and crossover probabilities [20]. To utilize such a GA for searching of Pareto-optimal mappings with a VFS configuration, we have to specify problem-specific GA attributes, namely a chromosome and fitness functions [20]. The chromosome is a representation of a GA solution (in our extended methodology a solution is a mapping with a VFS configuration) as a set of parameters (genes), joined into a string [20]. We represent mapping ${}^{n}A$, as a string of length $|A|$, where every gene is a CPU core $cpu_i \in cpu$ running at a frequency $f_{cp} \in f_{cpu}$. For a CPU core which offloads computations on a GPU, the gene also includes the GPU frequency $f_{gq} \in f_{gpu}$. An example of the chromosome, corresponding to the mapping with the VFS configuration, shown in Table 1, is given in Fig. 4.

4.3 Analytical Models as Fitness Functions

The aforementioned fitness functions are special functions that estimate the quality of the GA solutions and guide the GA-based search. We propose two analytical models and use them as fitness functions ϕ_1 and ϕ_2 during the GA-based search.

On the one hand, ϕ_1 estimates the system throughput during the GA search and is given as the following equation:

$$\phi_1 = 1/\tau \tag{1}$$

Note that our SDF model, for CNN inference, features pipeline execution of actors on CPUs to exploit both task-level and data-level parallelism (see Sect. 3.3). The bottleneck CPU in such pipeline execution will determine the system throughput $\phi_1 = 1/\tau$, where τ is the execution time needed for all SDF actors, mapped on the bottleneck CPU, to process one unit of data given as an input to the pipeline. So, we can compute τ as follows:

$$\tau = \max\{\max_{\forall cpu_i, i\in[m+1,n]}\{\tau_{cpu_i}\}, \max_{\forall cpu_i, i\in[1,m]}\{\tau_{cpu_i}{}'\}\} \tag{2}$$

where $\tau_{cpu_i}{}'$ and τ_{cpu_i} are the execution times needed for all SDF actors mapped on cpu_i to process one unit of data given as an input to the pipeline, when cpu_i offloads and does not offload tasks on a GPU, respectively. For every $cpu_i \in cpu$, τ_{cpu_i} or $\tau_{cpu_i}{}'$ is computed as:

$$\tau_{cpu_i} = \tau^t_{cpu_i} + \tau^{com}_{cpu_i} \tag{3}$$

$$\tau_{cpu_i}{}' = \tau^t_{cpu_i}{}' + \tau^{com}_{cpu_i} \tag{4}$$

where $\tau^t_{cpu_i}{}'$ and $\tau^t_{cpu_i}$ are the times cpu_i spends only on computations for all actors mapped on cpu_i, when cpu_i offloads and does not offload tasks on a GPU, respectively. $\tau^{com}_{cpu_i}$ is the time, spent by cpu_i, on communication with other embedded processors. $\tau^t_{cpu_i}$ and $\tau^t_{cpu_i}{}'$ are computed as:

$$\tau^t_{cpu_i} = \sum_{a_k \in {}^nA_i} \tau_{(F_k, cpu_i, f_{cp})} \tag{5}$$

$$\tau^t_{cpu_i}{}' = \sum_{a_k \in {}^nA_i} \tau_{(F_k, cpu_i, f_{cp}, f_{gq})} \tag{6}$$

where ${}^nA_i$ is the set of actors, mapped on cpu_i; F_k is the function of actor $a_k \in {}^nA_i$; $f_{cp} \in f_{cpu}$ is the frequency of cpu_i. $\tau_{(F_k, cpu_i, f_{cp})}$ is the time, taken by cpu_i to execute F_k at frequency f_{cp}; $\tau_{(F_k, cpu_i, f_{cp}, f_{gq})}$ is the time, taken by cpu_i at frequency f_{cp} to execute F_k, when the computation of F_k is offloaded on a GPU running at frequency $f_{gq} \in f_{gpu}$. The time $\tau^{com}_{cpu_i}$ is computed as:

$$\tau^{com}_{cpu_i} = \sum_{a_k \in {}^nA_i} (\tau_w(f_{cp}) * \sum_{c_{kj} \in C} U_{kj} + \tau_r(f_{cp}) * \sum_{c_{jk} \in C} V_{jk}) \tag{7}$$

where ${}^nA_i$ is the set of all actors, mapped on cpu_i; $c_{kj} \in C$ is an output channel of actor $a_k \in {}^nA_i$, to which, at each firing, actor a_k produces U_{kj} data tokens; $c_{jk} \in C$ is an input channel of actor a_k, from which, at each firing, actor a_k consumes V_{jk} data tokens; $\tau_r(f_{cp})$ and $\tau_w(f_{cp})$ specify the times, needed by a CPU core, to read and write one data token, at specific CPU frequency f_{cp}, respectively.

The accuracy of the analytical model ϕ_1 to estimate the system throughput depends on the accuracy of the parameter values $\tau_{(F_k, cpu_i, f_{cp})}$, $\tau_{(F_k, cpu_i, f_{cp}, f_{gq})}$,

$\tau_r(f_{cp})$ and $\tau_w(f_{cp})$. These values can be obtained accurately by real measurements on the target CPUs-GPUs MPSoC. An experimental confirmation of the accuracy of ϕ_1 is given in Sect. 5.2.

On the other hand, ϕ_2 estimates the system energy, consumed to process one unit of data given as an input to the system pipeline, during the GA search and is given as the following equation:

$$\phi_2 = \sum_{\forall cpu_i, i \in [m+1,n]} E_{cpu_i} + \sum_{\forall cpu_i, i \in [1,m]} E_{cpu_i}{}' + \sum_{\forall gpu_j \in gpu} E_{gpu_j} \tag{8}$$

where $E_{cpu_i}{}'$ and E_{cpu_i} are the energy consumption needed for all SDF actors, mapped on cpu_i, to process one unit of data given as an input to the pipeline, when cpu_i offloads and does not offload tasks on a GPU, respectively; E_{gpu_j} is the energy consumption needed for all offloaded SDF actors on gpu_j to process one unit of data given as an input to the pipeline. For every cpu_i and gpu_j, E_{cpu_i}, $E_{cpu_i}{}'$ and E_{gpu_j} are computed as:

$$\begin{aligned} E_{cpu_i} &= P_{idle}(cpu_i, f_{cp}) * T \\ &+ (P(cpu_i, f_{cp}, A) - P_{idle}(cpu_i, f_{cp})) * \tau_{cpu_i} \end{aligned} \tag{9}$$

$$\begin{aligned} E_{cpu_i}{}' &= P_{idle}(cpu_i, f_{cp}) * T \\ &+ (P(cpu_i, f_{cp}, A) - P_{idle}(cpu_i, f_{cp})) * \tau_{cpu_i}{}' \end{aligned} \tag{10}$$

$$\begin{aligned} E_{gpu_j} &= P_{idle}(gpu_j, f_{gq}) * T \\ &+ (P(gpu_j, f_{gq}, A) - P_{idle}(gpu_j, f_{gq})) * \tau_{gpu_j} \end{aligned} \tag{11}$$

where $P_{idle}(cpu_i, f_{cp})$ and $P_{idle}(gpu_j, f_{gq})$ are the power consumption of cpu_i and gpu_j, when there are no actors mapped on them, and they operate at frequencies f_{cp} and f_{gq}, respectively; $P(cpu_i, f_{cp}, A)$ and $P(gpu_j, f_{gq}, A)$ are the average power consumption of cpu_i and gpu_j when all actors of the SDF model (i.e., actor set A) are mapped on them, and they operate at frequencies f_{cp} and f_{gq}, respectively. T is the total time, taken by the system pipeline, to process one unit of data given as an input to the pipeline and is computed as follows:

$$T = \sum_{\forall cpu_i, i \in [m+1,n]} \tau_{cpu_i} + \sum_{\forall cpu_i, i \in [1,m]} \tau_{cpu_i}{}' \tag{12}$$

τ_{cpu_i} and $\tau_{cpu_i}{}'$ are calculated as shown in Eq. (3) and (4), respectively. τ_{gpu_j} is the time needed for gpu_j to execute all tasks offloaded by its corresponding cpu_i and is computed as follows:

$$\tau_{gpu_j} = \sum_{a_k \in {}^n A_j} \tau_{(F_k, gpu_j, f_{gq})} \tag{13}$$

where ${}^{n}A_j$ is the set of actors mapped on cpu_i and offloaded by cpu_i for execution on gpu_j; F_k is the function of actor $a_k \in {}^{n}A_j$; f_{gq} is the frequency of gpu_j. $\tau_{(F_k,gpu_j,f_{gq})}$ is the time, taken by gpu_j to execute F_k at frequency f_{gq}.

If no actors are mapped on cpu_i or gpu_j, then $\tau_{cpu_i}, \tau_{cpu_i}{}'$ or τ_{gpu_j} will be 0. In this case, E_{cpu_i}, $E_{cpu_i}{}'$ or E_{gpu_j} equals to the idle energy consumption, i.e., $P_{idle}(cpu_i, f_{cp}) * T$ or $P_{idle}(gpu_j, f_{gq}) * T$, as shown in Eq. (9), (10), (11).

The accuracy of the analytical model ϕ_2 to estimate the system energy consumption depends on the accuracy of the parameter values $P_{idle}(cpu_i, f_{cp})$, $P_{idle}(gpu_j, f_{gq})$, $P(cpu_i, f_{cp}, A)$ and $P(gpu_j, f_{gq}, A)$. These values can be obtained accurately by real measurements on the target CPUs-GPUs MPSoC. An experimental confirmation of the accuracy of ϕ_2 is given in Sect. 5.2.

5 Experimental Results

In this section, we present our results from experiments, in which real-world CNNs from the ONNX models zoo [21] are mapped and executed on the NVIDIA Jetson TX2 embedded CPUs-GPUs MPSoC [22]. The goal of the experiments is to demonstrate that, thanks to our contributions presented in this paper, our extended methodology can deliver CNN inference on embedded CPUs-GPUs MPSoCs with the same or higher throughtput but with lower energy consumption compared to existing DL frameworks that support CNN inference on such MPSoCs. First, we explain the setup for our experiments in Sect. 5.1. Then, in Sect. 5.2, we confirm the accuracy of our analytical models, introduced in Sect. 4.3. Finally, in Sect. 5.3, we use our extended methodology and models, introduced in Sect. 4, to find Pareto-optimal mappings and analyze our experimental results.

5.1 Experimental Setup

We use three real-world CNNs, namely Vgg19, Alexnet and Emotion_fer, from the ONNX models zoo [21] that take images as input for CNN inference. These CNNs are utilized in different applications and have diverse number and type of layers. Such diversity leads to a diverse scale of system throughtput and energy consumption when these CNNs are mapped and executed on the same hardware platform. Vgg19 and Alexnet are used for image classification and they have 19 layers and 8 layers, respectively. Emotion_fer is used for body, face, and gesture analysis and it has 10 layers. So, these three CNN models are sufficiently representative and good examples to apply our extended methodology on and to demonstrate its merits.

The three CNN models, mentioned above, are mapped and executed on the NVIDIA Jetson TX2 embedded CPUs-GPUs MPSoC [22] which features 6 CPUs (Quad-Core ARM and Dual-Core NVIDIA Denver 2) plus 1 Pa GPU device. The 6 CPUs are divided into two different clusters, where the CPUs from the same cluster can operate at 12 different frequencies and the GPU can operate at 13

different frequencies. We select NVIDIA Jetson TX2 as our experimental hardware platform because it is a well-known and easy-to-use embedded platform. Moreover, we can easily and accurately acquire the needed system throghput data and energy consumption data of each processor by setting timers within the executed code and by sampling the integrated power sensors onboard, respectively. In addition, NVIDIA Jetson TX2 is supported by the TensorRT framework [15], which is the best-known CNN deployment optimizer designed for NVIDIA embedded MPSoCs, as mentioned in Sect. 2. The results obtained by using our extended methodology are compared with TensorRT implementation results as reference in order to show the benefits of our extended methodology.

For every optimized reference system implementation, obtained by using TensorRT, the system throughput and energy consumption is directly measured on the NVIDIA Jetson TX2 platform, as the average value over 50 CNN inference executions. For the Pareto-optimal systems, obtained by using our extended methodology, the system throughput and energy consumption data is provided by our analytical models, introduced in Sect. 4.3. The two-objective GA of our methodology is executed with initial population size 5000, number of generations = 100, mutation probability = 5%. For all experiments, the original data precision (i.e., float32) is utilized in order to preserve the original CNN accuracy.

5.2 The Accuracy of Our Analytical Models

In this section, we confirm the accuracy of our system throughput and energy consumption analytical models, introduced in Sect. 4.3. We compare the estimated system throughput ϕ_1 and system energy consumption ϕ_2, obtained by our analytical models, with the corresponding numbers, obtained by direct measurements, on the reference system implementations, as described in Sect. 5.1. The results are shown in Table 2. In Column 1, we list the three experimental CNN models, mentioned in Sect. 5.1. For each CNN model, the experiments are performed with 9 different CPU and GPU frequency configurations. For the CPU and GPU frequencies, we use the maximum frequency, the minimum frequency, and a frequency in the middle, as shown in Column 2 and 3. For example, Row 2 shows that, when we perform the experiment on Vgg19 with CPU frequency 2.0 GHz and GPU frequency 1.3 GHz, we obtain system throughput of 14.30 img/s by a direct measurement, as shown in Column 4. Then, we obtain the estimated system throughput of 14.11 img/s by our analytical model, as shown in Column 5. Based on the data in Column 4 and 5, we calculate the error for the system throughput as $(14.11 - 14.30)/14.30 = -1.3\%$, shown in Column 6. In Column 6, a negative error value means that the system throughput is under-estimated and a positive value means that the system throughput is over-estimated. Similarly, Column 7 shows the system energy consumption of 0.58 J/img, obtained by a direct measurement and Column 8 shows the estimated system energy consumption of 0.56 J/img, obtained by our analytical model. Column 9 shows the error rate for the system energy consumption. We can see in Table 2 that the error rate for the system throughput is below 6% and the error rate for the energy consumption is below 9%. This fact confirms

that our analytical models are accurate enough for finding pareto optimal points during a complex design space exploration because such accuracy is sufficient to relatively compare different design points [23].

Table 2. Accuracy evaluation for our analytical models

CNN model	CPU frequency (GHz)	GPU frequency (GHz)	System throughput by measurement (img/s)	ϕ_1 (img/s)	Throughput error (%)	System energy consumption by measurement (J/img)	ϕ_2 (J/img)	Energy error (%)
Vgg19	2.00	1.30	14.30	14.11	−1.3	0.58	0.56	−3.4
	2.00	0.73	10.45	10.94	4.7	0.44	0.47	6.8
	2.00	0.11	1.89	1.96	3.7	1.32	1.27	−3.8
	1.27	1.30	13.04	12.67	−2.8	0.54	0.52	−3.7
	1.27	0.73	10.28	10.55	2.6	0.30	0.32	6.7
	1.27	0.11	1.88	1.91	1.6	0.59	0.62	5.1
	0.35	1.30	6.52	6.53	0.2	1.04	1.08	3.8
	0.35	0.73	6.23	6.55	5.1	0.44	0.42	−4.5
	0.35	0.11	1.85	1.79	−3.2	0.54	0.57	5.6
Alexnet	2.00	1.30	81.17	82.02	1.0	0.055	0.051	−7.3
	2.00	0.73	70.82	70.44	−0.5	0.049	0.048	−2.0
	2.00	0.11	13.47	13.56	0.7	0.148	0.159	7.4
	1.27	1.30	70.92	69.89	−1.5	0.054	0.055	1.9
	1.27	0.73	61.69	62.22	0.9	0.045	0.044	−2.2
	1.27	0.11	11.74	11.85	0.9	0.110	0.115	4.5
	0.35	1.30	38.88	38.69	−0.5	0.090	0.088	−2.2
	0.35	0.73	30.90	31.21	1.0	0.068	0.072	5.9
	0.35	0.11	6.46	6.50	0.6	0.155	0.161	3.9
Emotion_fer	2.00	1.30	224.7	220.5	−1.9	0.017	0.016	−5.9
	2.00	0.73	178.6	181.1	1.4	0.017	0.018	5.9
	2.00	0.11	35.2	35.9	2.0	0.062	0.059	−4.8
	1.27	1.30	192.3	189.3	−1.6	0.015	0.015	0
	1.27	0.73	164.7	166.8	1.3	0.012	0.013	8.3
	1.27	0.11	32.55	33.01	1.4	0.034	0.032	−6.3
	0.35	1.30	57.77	58.21	0.8	0.033	0.032	−3.0
	0.35	0.73	46.73	45.99	−1.6	0.023	0.024	4.3
	0.35	0.11	27.26	27.61	1.3	0.033	0.034	3.0

5.3 Pareto-optimal Mappings

In this section, we show the benefits of using our extended methodology, introduced in Sect. 4, through a comparison between the Pareto-optimal mappings, found by our methodology and the Pareto-optimal mappings, found by exhaustive search when using TensorRT for CNN inference implementation because TensorRT [15] is the best-known CNN deployment optimizer designed for NVIDIA embedded MPSoCs such as the NVIDIA Jetson TX2.

First, in order to find the Pareto-optimal mappings by using our methodology, we perform a design space exploration (DSE), by using the two-objective GA

of our methodology, introduced in Sect. 4.2, among possible mappings with a CPU and GPU VFS configuration, when Vgg19, Alexnet, and Emotion_fer are executed on the NVIDIA Jetson TX2 platform. The reason for using the two-objective GA for DSE is that the design space, which has to be explored and is supported by our methodology, consists of $|A|^{|cpu|*|gpu|*|f_{cpu}|*|f_{gpu}|}$ possible mappings. This is a huge number of mappings considering our experimental setup, thus exhaustive search is not feasible.

Second, in order to find the Pareto-optimal mappings when using TensorRT only, we perform a DSE by exhaustive search, among all possible mappings with a CPU and GPU VFS configuration, when Vgg19, Alexnet, and Emotion_fer are executed on the NVIDIA Jetson TX2 platform. Since TensorRT utilizes only one fixed CPU to offload all CNN inference tasks to one fixed GPU on NVIDIA Jetson TX2, the size of the design space when using only TensorRT depends on the possible CPU frequency levels $|f_{cpu}|$ and the possible GPU frequency levels $|f_{gpu}|$. Therefore, in this case, the design space consists of $|f_{cpu}| * |f_{gpu}| = 12 * 13 = 156$ design points to explore. Such small design space can be explored by exhaustive search in order to find all Pareto-optimal mappings with 100% guarantee.

Finally, we present and compare the Pareto-optimal mappings found by using the aforementioned two methods. The experimental results are shown in Fig. 5, 6 and 7. The horizontal axis shows the system throughput in images per second (img/s). The vertical axis shows the system energy consumption to process one image in joules per image (J/img). Each point in Fig. 5, 6 and 7 represents a Pareto-optimal mapping with certain system throughput and energy consumption. The red (+) points in the figures represent the Pareto-optimal mappings found by using our extended methodology. The green (×) points represent the Pareto-optimal mappings found by exhaustive search and using TensorRT only.

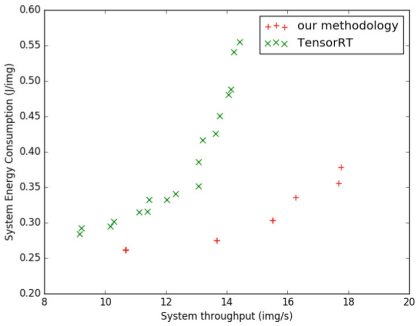

Fig. 5. Pareto-optimal mappings for Vgg19

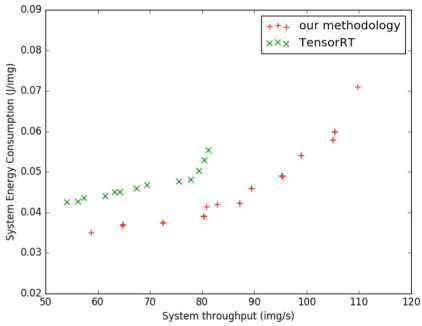

Fig. 6. Pareto-optimal mappings for Alexnet

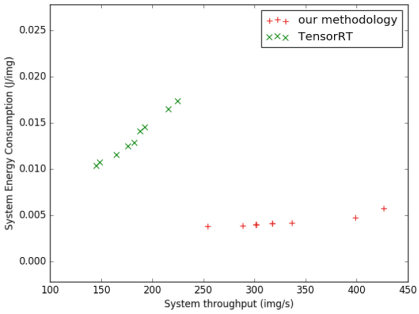

Fig. 7. Pareto-optimal mappings for Emotion_fer

From the experimental results, we can see that: (1) For Vgg19, as shown in Fig. 5, our methodology can deliver the same or better system throughput with a lower system energy consumption compared with TensorRT; (2) For Alexnet, as shown in Fig. 6, when the system throughput is lower than 100 img/s, our methodology can deliver the same or better system throughput with a lower system energy consumption compared with TensorRT. When the system throughput is higher than 100 img/s, only our methodology can deliver such system throughput but with a higher system energy consumption; (3) For Emotion_fer, as shown in Fig. 7, our methodology can always deliver a better system throughput with a lower system energy consumption compared with TensorRT. So, in conclusion, our extended methodology is able to achieve both energy efficiency and high throughput when deploying CNN models on embedded CPUs-GPUs MPSoCs.

6 Conclusions

In this paper, we propose an extended methodology to achieve energy efficiency and high throughput when deploying CNN models on embedded CPUs-GPUs MPSoCs. Our methodology finds Pareto-optimal mappings of a CNN model onto a CPUs-GPUs MPSoCs with VFS configurations with the help of a two-objective GA which optimizes the system throughput and energy consumption simultaneously. Moreover, we propose two analytical models, that are used as fitness functions in the two-objective GA to evaluate very fast the system throughput and energy consumption of CNNs mapped onto embedded CPUs-GPUs MPSoCs and we confirm the high accuracy of these two analytical models by experimental evidence. Finally, the experimental results of real-world CNNs execution on the NVIDIA Jetson TX2 platform show that, compared with the best-known CNN deployment optimizer TensorRT, our extended methodology is able to achieve both energy efficiency and high throughput when deploying CNN models on embedded CPUs-GPUs MPSoCs.

References

1. Alom, Md.Z., et al. The history began from Alexnet: a comprehensive survey on deep learning approaches. arXiv preprint arXiv:1803.01164 (2018)
2. Diamant, A., et al.: Deep learning in head & neck cancer outcome prediction. Sci. Rep. **9**(1), 1–10 (2019)
3. Do, T.-D., et al.: Real-time self-driving car navigation using deep neural network. In: 2018 4th International Conference on Green Technology and Sustainable Development (GTSD), pp. 7–12. IEEE (2018)
4. Alexey A Shvets et al. Automatic instrument segmentation in robot-assisted surgery using deep learning. In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 624–628. IEEE (2018)
5. Martin, G.: Overview of the MPSOC design challenge. In 2006 43rd ACM/IEEE Design Automation Conference, pp. 274–279. IEEE (2006)
6. Wang, S., et al.: High-throughput CNN inference on embedded arm big little multicore processors. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. **39**, 2254–2267 (2019)
7. Linpeng Tang et al. Scheduling computation graphs of deep learning models on manycore cpus. arXiv preprint arXiv:1807.09667 (2018)
8. Abadi, M., et al.: Tensorflow: large-scale machine learning on heterogeneous systems (2015)
9. Jia, Y., et al.: Caffe: convolutional architecture for fast feature embedding. In: Proceedings of the 22nd ACM International Conference on Multimedia, pp. 675–678 (2014)
10. Parvat, A., et al.: A survey of deep-learning frameworks. In 2017 International Conference on Inventive Systems and Control (ICISC), pp. 1–7. IEEE (2017)
11. Song, L., et al.: Hypar: towards hybrid parallelism for deep learning accelerator array. In: 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 56–68. IEEE (2019)
12. Kang, D., et al.: C-good: C-code generation framework for optimized on-device deep learning. In: Proceedings of the International Conference on Computer-Aided Design, pp. 1–8 (2018)
13. Huynh, L.N., et al.: Deepsense: a GPU-based deep convolutional neural network framework on commodity mobile devices. In: Proceedings of the 2016 Workshop on Wearable Systems and Applications, pp. 25–30 (2016)
14. Huynh, L.N., et al.: Deepmon: mobile GPU-based deep learning framework for continuous vision applications. In: Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services, pp. 82–95 (2017)
15. Nvidia tensorrt framework. https://developer.nvidia.com/tensorrt
16. Minakova, S., Tang, E., Stefanov, T.: Combining task- and data-level parallelism for high-throughput CNN inference on embedded CPUs-GPUs mpsocs. In: 20th International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS 2020), July 05–09 (2020)
17. Lee, E.A., Messerschmitt, D.G.: Static scheduling of synchronous data flow programs for digital signal processing. IEEE Trans. Comput. **100**(1), 24–35 (1987)
18. Bilsen, G., et al.: Cycle-static dataflow. IEEE Trans. Signal Process. **44**(2), 397–408 (1996)
19. Deb, K., Gupta, H.: Searching for robust pareto-optimal solutions in multi-objective optimization. In: Coello Coello, C.A., Hernández Aguirre, A., Zitzler, E. (eds.) EMO 2005. LNCS, vol. 3410, pp. 150–164. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31880-4_11

20. Sastry, K., et al.: Genetic algorithms. In: Search Methodologies, pp. 97–125. Springer, Heidelberg (2005). https://doi.org/10.1007/3-540-29623-9_7150
21. Onnx models zoo. https://github.com/onnx/models
22. Nvidia Jetson TX2. https://developer.nvidia.com/embedded/jetson-tx2
23. Palesi, M., Givargis, T.: Multi-objective design space exploration using genetic algorithms. In: The Tenth International Symposium on Hardware/Software codesign, pp. 67–72 (2002)