On the Task Mapping and Scheduling for DAG-based Embedded Vision Applications on Heterogeneous Multi/Many-core Architectures

Stefano Aldegheri
Department of Computer Science
University of Verona
stefano.aldegheri.mail@gmail.com

Nicola Bombieri
Department of Computer Science
University of Verona
nicola.bombieri@univr.it

Hiren Patel
Electrical and Computer Engineering
University of Waterloo
hdpatel@uwaterloo.ca

***Abstract*—In this work, we show that applying the heterogeneous earliest finish time (HEFT) heuristic for the task scheduling of embedded vision applications can improve the system performance up to 70% w.r.t. the scheduling solutions at the state of the art. We propose an algorithm called exclusive earliest finish time (XEFT) that introduces the notion of exclusive overlap between application primitives to improve the load balancing. We show that XEFT can improve the system performance up to 33% over HEFT, and 82% over the state of the art approaches. We present the results on different benchmarks, including a real-world localization and mapping application (ORB-SLAM) combined with the NVIDIA object detection application based on deep-learning.**

***Index Terms*—Embedded vision applications, static mapping and scheduling, OpenVX, heterogeneous architectures.**

I. Introduction

Deploying computer vision applications on embedded systems is a pervasive trend across many different fields, from autonomous driving to robotics and security [1]. In this context, OpenVX [2] has been proposed to help the development of computer vision applications, and it is increasingly considered the standard for application design and system-level optimization in the embedded vision community.

Prior research efforts have used OpenVX for embedded vision [3], [4], and attempted to optimize the performance of the generated code. They proposed techniques to implement different data access patterns such as DAG *node merge*, *data tiling*, and parallelization via OpenMP. There also have been efforts to make the OpenVX task scheduling deliver real-time guarantees [5]. Nevertheless, to the best of our knowledge, there is no prior work that focuses on efficient mapping strategies and its corresponding scheduling of OpenVX (DAG-based) applications for heterogeneous architectures. Prior approaches that propose mapping strategies for OpenVX considered each DAG node to have only one exclusive implementation (e.g., either GPU or CPU), and the mapping is driven by the availability of the node's implementation in the library: If a node has a GPU implementation then it is mapped on the GPU. Otherwise it is mapped on a CPU core.

To take into consideration the heterogeneity of the target architectures, the possible multiple implementations of DAG nodes, and the problem complexity, we first propose an implementation of the heterogeneous earliest finish time (HEFT) heuristic [6] for *static* mapping and scheduling of OpenVX applications. We show that the HEFT implementation sensibly outperforms (i.e., up to 70% of performance gain) the state-of-the-art solution currently adopted in one of the most widespread embedded vision systems (i.e., NVIDIA VisionWorks on NVIDIA Jetson TX2). Then, we show that such a heuristic, when applied to DAG graphs for which *not every* node has multiple implementations, can lead to idle periods for the computing elements (CEs). Since not having multiple implementations for all nodes happens in a majority of real embedded vision contexts, we propose an algorithm that reorganizes the HEFT ranking to improve the load balancing. The algorithm aims at generating sequences of nodes with the single implementation in the ranking with the objective of reducing idle times caused by the combination of DAG constraints and exclusive implementation.

We present the results obtained on a large set of synthetic benchmarks and on a real-world localization and mapping application (ORB-SLAM) combined with an NVIDIA image recognition application based on deep-learning.

The paper is organized as follows. Section II presents the background and the related work. Section III presents the proposed HEFT and XEFT implementations. Sections IV and V present the results and the concluding remarks.

II. Background and Related Work

An extensive overview of mapping and scheduling strategies can be found in [7]. Considering our target applications (i.e., embedded vision), in this work we limit our focus on the class of static scheduling for heterogeneous architectures, for which we summarize the most recent and related works next.

TETRiS [8] is a run-time system for static mapping of *multiple applications* on heterogeneous architectures. It does not apply to DAG-based applications, and it does not support the concept of *multiple* (i.e., one implementation of a node for each CE) and *exclusive* implementations (i.e., the implementation of a node for a given CE) of nodes for heterogeneous CEs (e.g., CPUs and GPUs).

We consider an application being represented by a directed acyclic graph (DAG), $G = (V, E)$, where V is the set of v tasks and E is the set of e edges between the tasks (we use the terms *task* and *node* interchangeably in the paper). Each

This research work has been partially supported by the project "Dipartimenti di Eccellenza 2018-2022" funded by the Italian Ministry of Education, Universities and Research (MIUR).

edge $(t, q) \in E$ represents the precedence constraint such that task t should complete its execution before task q starts.

In [9] first, and then in [5], the authors proposed an approach and its optimization to schedule DAG-based OpenVX applications for multi-cores and GPU architectures. Their approach allowed the application performance to be increased by overlapping sequential executions of the application. On the other hand, it does not consider the multiple implementations of DAG nodes, i.e., the mapping algorithm targets the best *local* solution: If there exists a GPU kernel for a DAG node then that node is mapped onto the GPU.

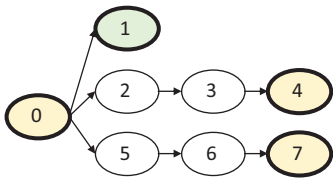
To support the mapping of each DAG node onto one CE among different heterogeneous possibilities we consider the HEFT algorithm [6]. We propose an implementation of the task mapping and scheduling for OpenVX, which is based on the HEFT heuristic with up-word ranking and max functions [6]. To our knowledge, the application of HEFT on such platforms has not been considered in prior work. We adapted the algorithm in order to support the exclusive implementation of DAG nodes. We then propose an optimization of HEFT that, starting from a given ranking list, it reorganizes the list to improve the task overlapping. It is important to note that the proposed optimization is independent of the HEFT variants, since it applies to any ranking list. For the sake of clarity and without loss of generality, we assume hardware heterogeneity as the combination of CPU cores with one accelerator (i.e., GPU). The concepts and the algorithm can be extended to other heterogeneous architectures, in which more accelerators are combined with the CPU cores.

III. Methodology

We illustrate our methodology with the example in Fig. 1, which represents the DAG model of an application to be deployed on a heterogeneous CPU/GPU board, and for which there exists a library of primitives for the node implementations. The library provides the *exclusive* implementation for CPU of node #0 (which is the application starting point), of node #4 and of node #7, the exclusive implementation for GPUs (i.e., GPU kernel) of node #1, while it provides the multiple implementations (CPU implementation and an equivalent GPU kernel) of nodes #2, #3, #5, and #6. The table in Fig. 1 summarizes the execution time of each primitive when executed in isolation on the corresponding CEs.

For brevity, we assume the CPU-GPU data transfer time to be negligible in this example (it has been considered in the methodology and in our experimental analysis), and we consider the heterogeneous target system to consist of one CPU core and one GPU.

Fig. 2(a) represents the task mapping and scheduling of the application implemented by the NVIDIA VisionWorks runtime system. A similar approach is implemented by the AMD OpenVX (AMDOVX) runtime system. The mapping relies on the best *local optimization*, that is, a node is mapped on the GPU if there exists the corresponding GPU kernel in the library. The scheduling relies on the topological order of tasks in the DAG, and honours the topological order constraints among nodes. However, this approach does not implement overlapping among tasks.



node #	t_{CPU} (ms)	t_{GPU} (ms)	HEFT rank (max)
0	1	-	15
1	-	10	10
2	4	2	14
3	4	1	10
4	6	-	6
5	4	2	14
6	4	1	10
7	6	-	6

Fig. 1: Example of DAG, execution time of tasks mapped on CPU/GPU, and the corresponding HEFT ranking.

Fig. 2(b) shows the scheduling of the proposed HEFT implementation for OpenVX, which takes advantage of both task overlapping and the mapping optimized at *system-level*. Starting from a task ranking generated as described in equation (1), one node at a time is mapped onto the CE that involves a better *system* execution time. This means that a node can be mapped on a CE that leads to a higher execution time at task level (see tasks of nodes #5, #3, and #6 in Fig. 2(b)). Assuming that the nodes have the multiple implementations and not necessarily all the GPU kernels outperform the corresponding CPU primitives, the HEFT algorithm heuristically provides load balancing on the CEs by *overlapping* the task execution. This leads to a reduction of the application makespan. However, there are nodes that do not have multiple implementations. In this case, the iterative nature of task mapping and balancing of HEFT can lead to large idle periods. An example is the idle period on the GPU in Fig. 2(b), which could be avoided (or reduced) by an implementation of node #7 for GPU.

In general, the main limitation of HEFT in the context of heterogeneous architectures is that, by following the rank order, it maps one task at a time by guaranteeing the best load balancing at each iteration. It does not consider the single or multiple implementations of the nodes.

Our idea is that the load balancing can be improved by prioritizing the overlapping between exclusive nodes, which we call *exclusive overlapping*. Considering the standard definition of *overlapping* between two tasks t and q:

$$O(t, q) = \max\left(0, \min\left(t_{end}, q_{end}\right) - \max\left(t_{start}, q_{start}\right)\right), \quad (1)$$

where t_{start} and t_{end} are the starting and ending times of t, respectively. We define *exclusive overlapping (XO)* between two tasks t and q running on different CEs as follows:

$$XO(t, q) = \begin{cases} O(t, q), & \text{if } (\nexists\ t_{CPU} \wedge \nexists\ q_{GPU}) \\ & \vee\ (\nexists\ t_{GPU} \wedge \nexists\ q_{CPU}) \\ 0, & \text{otherwise} \end{cases}, \quad (2)$$

where t_{CPU} (t_{GPU}) represents the CPU implementation (GPU implementation) of task t.

Exclusive overlapping applies to nodes that cannot compete for the same CE due to exclusive implementations. We define the total overlapping and the total exclusive overlapping between tasks *of an application* A as follows:

$$O(A) = \sum_{t,q \in A} O(t, q), \quad (3)$$

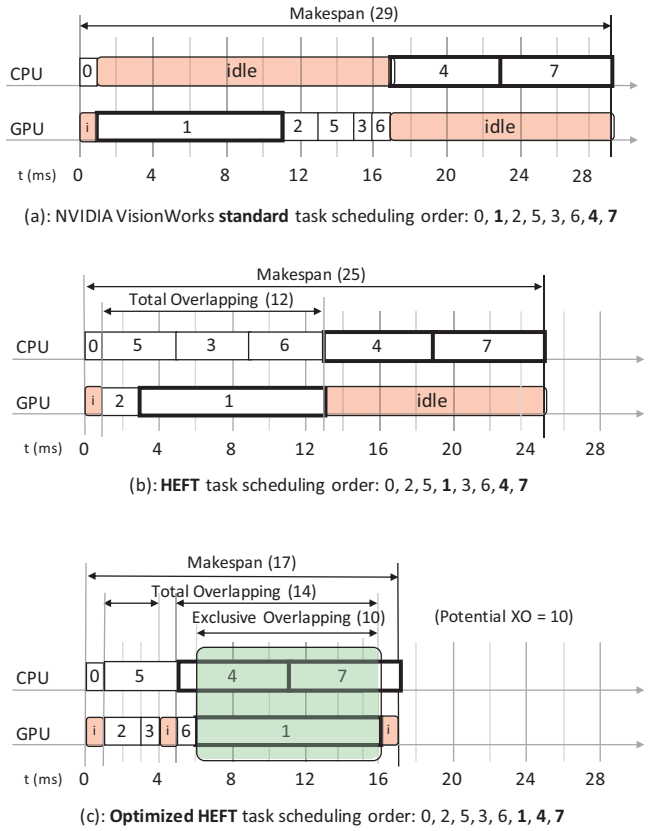


Fig. 2: Task scheduling algorithms of the DAG of Fig. 1: native NVIDIA VisionWorks (a), HEFT (b), and the proposed optimized HEFT (c).

$$XO(A) = \sum_{t,q \in A} XO(t,q), \tag{4}$$

Fig. 2(c) shows the exclusive overlapping between the tasks of the example. The main idea is that increasing XO can reduce idle times caused by the combination of DAG constraints and exclusive implementation (e.g., the idle time on the GPU between instants 13 and 25 in Fig. 2(b)). Our experimental results show that increasing XO corresponds to an increase of the standard overlapping and to an improvement of performance.

To increase XO, we propose an algorithm (*Algorithm* 1) that, starting from a given ranking list, it reorganizes the list to identify and generate *clusters* of exclusive nodes, i.e., sequences of exclusive nodes that are strictly consecutive in the ranking (see nodes #1, #4, and #7 in Fig. 2(c)).

The algorithm starts by defining the standard HEFT ranking from the application graph (row 2). One node at a time, and for all nodes of the ranking (row 4), the algorithm identifies a new cluster starting from the next exclusive node of the list (row 6). It searches among all the next nodes in the ranking that are exclusive and that do not have topological constraints in the DAG with the current cluster nodes (i.e., that are not in the same DAG path). The second condition is necessary to avoid serialization among the cluster nodes. The algorithm calculates the total makespan of the cluster nodes for each CE

Algorithm 1 Cluster identification and generation

```
1:  procedure BUILDCLUSTER(graph)
2:      rank ← build_rank(graph)
3:      i ← 0
4:      while i < size(rank) do
5:          if rank[i] is exclusive then
6:              candidates ← rank[i]
7:              j ← i + 1
8:              while j < size(rank) do
9:                  if rank[j] is exclusive ∧∀p ∈ candidates, p ↛ rank[j] then
10:                     candidates ← candidates ∪ rank[j]
11:                 j ← j + 1
12:             total_cpu ← reduce_sum(candidates, t_cpu)
13:             total_gpu ← reduce_sum(candidates, t_gpu)
14:             C ← exclusive CPU nodes in candidates
15:             G ← exclusive GPU nodes in candidates
16:             if total_gpu < total_cpu / n_cores then
17:                 cluster ← G
18:                 for all c ∈ C do
19:                     t ← reduce_sum(cluster, t_cpu)
20:                     if |total_gpu − (t+c_cpu)/n_cores| < |total_gpu − t/n_cores| then
21:                         cluster ← cluster ∪ c
22:             else
23:                 for all g ∈ G do
24:                     t ← reduce_sum(cluster, t_gpu)
25:                     if |total_cpu/n_cores − t + g_gpu| < |total_cpu/n_cores − t| then
26:                         cluster ← cluster ∪ g
27:             APPLY(rank, cluster)
28:             i ← cluster_end + 1
29:         else
30:             i ← i + 1
        return rank
```

(rows 12 and 13). The shortest makespan characterizes the maximum XO of the cluster under generation. For the sake of clarity, in *Algorithm* 1, we considered two possible cluster makespans ($total_{cpu}$ and $total_{gpu}$). The algorithm completes the cluster identification by including all nodes (for the same CE) that give the shortest makespan and, incrementally, with the exclusive nodes that bring to a comparable makespan on the other CEs (rows 16-26). The first node of any other CE that causes makespan unbalancing starts a new cluster in the following iteration of the algorithm.

The algorithm implements the cluster generation (`APPLY(rank,cluster)`) by moving either the identified nodes up on the ranking or the cluster down on the ranking. All the identified nodes (i.e., *candidates* in *Algorithm* 1) and the cluster can be moved and made adjacent since, for the condition in row 9, they cannot have topological constraints against each other.

IV. Experimental results

We evaluated the proposed algorithms by considering two categories of benchmarks. The first is a mapping and localization application (ORB-SLAM) [10] combined with an image recognition system based on Deep Learning (DL) [11]. Such a real-world computer vision application implements the simultaneous localization and mapping problem when one or more RGB camera sensors are adopted. It computes, in real-time, the camera(s) trajectory, a sparse 3D reconstruction of the scene, and car recognition through DL. We considered three different versions of the applications: *Monocular* with a 41 node DAG, *stereo* (81 nodes), and *4-stereo* (161 nodes). We used the standard KITTI input dataset [12] for the evaluation.

The second category is a set of 40,000 synthetic DAGs, with different characteristics: Size (from 20 to 250 nodes), node degree, execution times of CPU and GPU node implementations, exclusive vs. multiple implementations of nodes,

ORB-SLAM version (#CPU cores)	Max XO (ms)	Overlap (ms)		XO (ms)		Idle time (%)		XO/Max_XO (%)		Makespan (ms)			Speedup			SLR	
		HEFT	XEFT	HEFT	XEFT	HEFT	XEFT	HEFT	XEFT	VW	HEFT	XEFT	HEFT on VW	XEFT on VW	XEFT on HEFT	HEFT	XEFT
Monocular(2)	44.1	60.3	70.4	0.3	36.8	42.9%	14.4%	0.7%	83.5%	84.7	52.8	35.3	37.7%	58.3%	33.2%	2.01	1.34
Monocular(3)	44.1	84.9	92.0	16.6	34.2	29.3%	25.1%	37.6%	77.4%	84.7	40.0	34.4	52.8%	59.5%	14.1%	1.52	1.31
Monocular(4)	44.1	107.3	109.4	32.9	37.4	20.2%	31.2%	74.7%	84.7%	84.7	33.6	32.9	60.3%	61.2%	2.2%	1.28	1.25
Monocular(5)	44.1	119.2	119.2	42.3	42.3	22.0%	22.0%	95.9%	95.9%	84.7	30.6	30.6	63.9%	63.9%	0.0%	1.16	1.16
Stereo(2)	50.0	81.0	87.0	0.0	36.7	51.2%	31.7%	0.0%	73.4%	128.8	83.0	63.7	35.6%	50.5%	23.2%	3.16	2.43
Stereo(3)	75.0	110.8	123.1	0.0	57.3	44.2%	10.0%	0.0%	76.4%	128.8	66.2	45.6	48.6%	64.6%	31.1%	2.52	1.74
Stereo(4)	88.2	134.1	147.1	6.2	50.6	40.7%	28.1%	7.0%	57.3%	128.8	56.6	44.2	56.1%	65.7%	21.9%	2.16	1.68
Stereo(5)	88.2	163.4	167.8	13.0	44.9	33.4%	36.4%	14.8%	50.9%	128.8	49.1	43.3	61.9%	66.4%	11.8%	1.87	1.65
4-stereo(2)	50.0	130.7	137.3	0.0	33.4	54.6%	43.9%	0.0%	66.8%	217.0	144.0	122.4	33.7%	43.6%	15.0%	5.48	4.66
4-stereo(3)	75.0	181.4	188.8	0.0	27.1	45.1%	32.9%	0.0%	36.1%	217.0	110.1	93.8	49.3%	56.8%	14.8%	4.20	3.57
4-stereo(4)	100.0	222.4	236.0	3.8	89.0	40.3%	22.9%	3.8%	89.0%	217.0	93.2	76.6	57.1%	64.7%	17.8%	3.55	2.92
4-stereo(5)	125.0	258.3	276.5	8.1	114.4	38.1%	17.4%	6.4%	91.5%	217.0	83.5	66.9	61.5%	69.2%	19.8%	3.18	2.55

TABLE I: Experimental results with ORB-SLAM+DL on Jetson TX2

CPU/GPU speedup for nodes with multiple implementations.

We used the NVIDIA Jetson TX2 as target architecture. It consists of a dual-core Denver2 64-bit CPU + quad-core ARM A57 CPU, and a 256-core Pascal GPU. We evaluated the embedding, mapping and scheduling process for 2, 3, 4, and 5 CPUs core + 1 GPU. We assigned one CPU core for the OpenVX runtime system and CPU/GPU synchronization.

Table I presents the results obtained by running the different configurations of ORB-SLAM+DL on the target architecture with the different CPU/GPU scenarios (i.e., #CPU cores enabled beside the GPU). The table shows the comparison of the different mapping and scheduling approaches, i.e., NVIDIA Vision Work (VW) [13], standard HEFT, and the optimized HEFT (XEFT). The schedule length ratio (SLR) normalizes the makespan over the maximum critical path.

Our results show that, as expected, HEFT sensibly improves the application performance w.r.t. the scheduling system currently released with the NVIDIA Vision Work library. The improvement ranges from a minimum of 33.7% to a maximum of 69.2%. Then, the table shows that XEFT provides an exclusive overlapping degree that is higher than that provided by HEFT in almost all cases (see double column XO). The only case of XO reduction is with the simpler application versions (monocular) run on a large number of CPU cores (i.e., 5). For this reason, in these two contexts, also the overall performance improvement provided by XEFT against HEFT is slightly negative (-5.7% and -3.8%).

The idle time values show that this category of benchmarks scheduled with HEFT suffers from load imbalance. The efficiency of HEFT to provide exclusive overlapping is reported in column XO/Max_XO and it is higher with simpler applications and with low levels of maximum potential XO. The clustering effect of XEFT is a reduction of the idle times in the CEs and an increase of the XO efficiency. The two values improve by increasing the application complexity.

In general, XEFT provides a performance improvement w.r.t. HEFT up to 33.2% and, more importantly, it provides the same or better performance of HEFT with less architectural resources (e.g., with one less CPU core).

With the synthetic benchmarks, the results underline that XEFT generally outperforms HEFT with benchmarks for which (i) HEFT suffers from idle time, and (ii) the XO efficiency of HEFT is low. XEFT cannot improve the HEFT performance if the benchmark, with HEFT, is already well balanced (low idle time) or already presents high XO efficiency. In these cases, XEFT can decrease the performance up to 19%.

V. Conclusion and future work

We analysed the HEFT heuristic limitation when applied to actual embedded vision applications. We proposed an algorithm called XEFT that reorganizes the HEFT rank by exploiting the concept of exclusive overlapping. We conducted the experimental analysis on a large set of synthetic benchmarks and on a combination of computer vision and deep learning applications to understand the correlation of the application characteristics and the efficiency of the proposed approach.

References

[1] Embedded Vision Alliance, "Applications for Embedded Vision." https://www.embedded-vision.com/applications-embedded-vision.

[2] Khronos Group, "OpenVX: Portable, Power-efficient Vision Processing." https://www.khronos.org/openvx.

[3] G. Tagliavini, G. Haugou, A. Marongiu, and L. Benini, "ADRENALINE: An OpenVX Environment to Optimize Embedded Vision Applications on Many-core Accelerators," in *Proc. of IEEE International Symposium on Embedded Multicore/Many-core SoCs*, pp. 289–296, 2015.

[4] S. Aldegheri and N. Bombieri, "Extending OpenVX for model-based design of embedded vision applications," in *Proc. of IEEE International Conference on VLSI and System-on-Chip, VLSI-SoC*, pp. 1–6, 2017.

[5] M. Yang, T. Amert, K. Yang, N. Otterness, J. Anderson, F. Smith, and S. Wang, "Making OpenVX Really 'Real Time'," in *Proc of IEEE Real-Time Systems Symposium (RTSS)*, pp. 80–93, 2019.

[6] H. T. et al., "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[7] A. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *Proc. od ACM/IEEE Design Automation Conference*, 2013.

[8] A. Goens, R. Khasanov, J. Castrillon, M. Hähnel, T. Smejkal, and H. Härtig, "TETRiS: A Multi-Application Run-Time System for Predictable Execution of Static Mappings," in *Proc. of ACM Int. Workshop on Software and Compilers for Embedded Systems*, pp. 11–20, 2017.

[9] G. Elliott, K. Yang, and J. Anderson, "Supporting Real-Time Computer Vision Workloads Using OpenVX on Multicore+GPU Platforms," in *Proceedings - Real-Time Systems Symposium*, pp. 273–284, 2016.

[10] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[11] NVIDIA, "DL inference networks and deep vision primitives with TensorRT and NVIDIA Jetson." https://github.com/dusty-nv/jetson-inference.

[12] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research*, 2013.

[13] NVIDIA Inc., "VisionWorks." https://developer.nvidia.com/embedded/visionworks.