

Formulation and Practical Solution for the Optimization of Memory Accesses in Embedded Vision Systems

Khadija HADJ SALEM

LCIS Laboratory

50 rue Barthélémy de Laffemas BP 54
 26902 Valence, France

khadija.hadj-salem@lcis.grenoble-inp.fr

Yann KIEFFER

LCIS Laboratory

50 rue Barthélémy de Laffemas BP 54
 26902 Valence, France

yann.kieffer@lcis.grenoble-inp.fr

Stéphane MANCINI

TIMA Laboratory

46 avenue Félix Viallet
 38031 Grenoble, France

stephane.mancini@imag.fr

Abstract—The design of embedded vision systems carries a difficult challenge regarding the access times of memories holding image data for some particular cases of image treatments. This paper studies the optimization challenge reflecting the efficient operation of adhoc memory systems proposed by electronic designers to alleviate this problem. New algorithms are proposed for producing solutions to this 3-objective problem, and numerical experiments are conducted on real-world data for validating their efficiency.

I. INTRODUCTION

The design of embedded vision systems carries many challenges, one of which is the efficient access to the image memory. An architectural solution was proposed by Mancini and al. [1] in the form of a software tool that creates an ad-hoc memory hierarchy for non-linear image accesses. But operating this kind of systems is itself an optimization challenge, involving 3 objectives reflecting 3 main electronic design parameters. To the best of our knowledge, this problem has not been studied before in the optimization literature.

The remaining of this paper is organized as follows. After describing the Memory Management Optimization design software, we explain the related optimization problematic set by the efficient operation of the circuits produced by this tool, and give a formal multi-objective mathematical model for this problem, as well as several sub-problems of interest. We then review the state of the art. After giving lower bounds for the 3-objective problem, we analyze the complexity of some of the mono-objective sub-problems. The description of new approaches, including a simple heuristic and two algorithms, is then given. Numerical experiments follow, which are conducted on real-world data in order to validate their efficiency, and a conclusion and perspectives section closes the paper.

II. EMBEDDED VISION SYSTEMS CONTEXT: ARCHITECTURAL SOLUTION AND OPTIMIZATION PROBLEMATIC

A. Memory Management Optimization Tool

Among modern-day electronic devices, embedded vision systems such as picture and video cameras represent a specific

design challenge with respect to memory management. Image sizes are measured in 100's of kbs or even Mbs, while the access times must be short enough to allow the quick handling of the data. For example, a live video feed may have 30 frames per second, meaning that the handling of one image (frame) must take less than 1/30 s. But it is a well-known fact in electronic memory design that access times grow with the size of the memory to be accessed. Due to this fact, it is not possible to reach the performance needed with the simple use of memories. Something has to be added to improve the access times.

For digital image treatments (also called *kernels*) that have linear access patterns to the memory addresses, usual caches as used for CPUs will solve this problem. But for non-linear access patterns, the problem remains a big hurdle for the easy and efficient design of kernel circuit designs. An illustration of a non-linear kernel is given in Fig. 1.

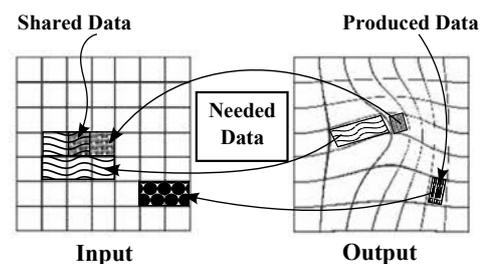


Fig. 1. Example of a non-linear kernel

To address this problematic, Mancini and al. [1] have designed a software generator of memory hierarchies tailored to one particular non-linear kernel. Their solution, called *Memory Management Optimization (MMOpt)*, takes as input a non-linear kernel for which the memory hierarchy is to be produced, such as the one shown in Fig. 1; it analyzes its access patterns; it then designs a run-time behavior for the whole resulting block *Tile Processing Unit (TPU)*; and it finally outputs the design of the TPU, together with the

information needed to orchestrate its operational behavior.

We give some details about the architecture of the TPU, as shown in Fig. 2. It is made of (a) a *Prefetching Unit* (PU) that loads data from external memory to local buffers, and (b) a *Processing Engine* (PE), that computes output data using prefetched input ones.

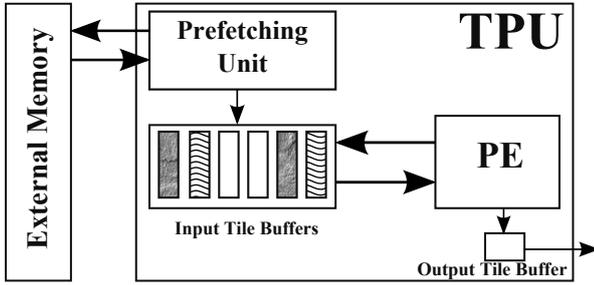


Fig. 2. Architecture template of the TPU

MMOpt computes and encodes into the TPU a schedule of prefetches and a schedule of computations. Hence memory accesses in the final system are deterministic (i.e. independent of pixel values), and this is a requirement of the input kernel for the whole MMOpt scheme to work out.

B. Optimization for MMOpt

When designing electronic circuits, some of the important design criteria are the area of the circuit produced, since it is directly related to production costs; the energy consumption, which may be limited, and which conditions the battery life for battery-powered devices; and the performance, which is usually a design parameter reflecting reactivity, and fluidity in the case of moving images.

TPUs produced by MMOpt embed schedules for the prefetches of input tiles and computations of output tiles (see Fig. 3).

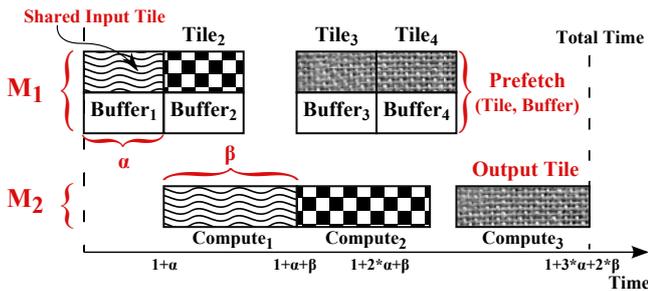


Fig. 3. Prefetches and computations schedules

The architecture of the TPU and those schedules will impact those three design parameters in the following way: the number of buffers of the TPU will account for most of its area; the number of prefetches reflects the main part of the energy consumption; and the performance is related to the completion time of the whole prefetches-computation schedule for the computation of one image.

Since MMOpt is a fully automatic electronic design software, computing good schedules is both a necessity and an opportunity for the circuit designers to deliver, with the help of MMOpt, low-cost, low-energy and efficient TPUs.

III. THE 3-OBJECTIVE PROCESS SCHEDULING AND DATA PREFETCHING PROBLEM

As you may know, the Integer Linear Programming formulation is used in the operations research as a modeling approach for the optimization problems. In this study, we have chosen to formulate our optimization issue by an off-line, 3-objective model with clearly delineated inputs and outputs, which we now present. This multi-objective mathematical formulation is a very flexible modeling approach that allows a preciseness for dealing with many specific sub-problems.

A. Problem Statement and Assumptions

The main multi-objective optimization problem considered in this paper is called *3-objective Process Scheduling and Data Prefetching Problem* (3-PSDPP). It involves the definition of the number of buffers of the TPU, the scheduling of output tiles computations, and the scheduling of input tiles prefetches, while respecting a requirement constraint between prefetches and computations.

The main assumptions that apply to the 3-PSDPP are the following:

- 1) Input tile sizes are identical and each input tile fits exactly into one buffer.
- 2) There is no distinction between buffers, i.e. any input tile may be prefetched into any buffer.
- 3) All input (respectively output) tiles and the subset of input tiles required to compute each output tile are known in advance.
- 4) Only one input (output) tile can be prefetched (computed) at a time.
- 5) The prefetch operations and the computation steps may be carried out simultaneously.
- 6) Input (output) tile prefetch (respectively computation) times are constant and identical.

B. Formulation for 3-PSDPP

The 3-PSDPP problem consists of finding an appropriate computation sequence in which output tile will be computed, and an associated prefetch sequence in which input tile will be prefetched from the external memory to the buffers, that simultaneously minimizes the number of prefetches, the number of buffers, and the completion time. We now describe the input data of our problem 3-PSDPP, the expected output, the constraints, and the 3 formal objectives reflecting the 3 electronic design parameters (Mathematical Formulation given in Table I).

1) *Inputs*: a 3-PSDPP instance is represented by a 5-tuple $(\mathcal{X}, \mathcal{Y}, (\mathcal{R}_y)_{y \in \mathcal{Y}}, \alpha, \beta)$ where \mathcal{X} is the set of input tiles to be prefetched, and \mathcal{Y} is the set of output tiles to be computed. Each output tile y requires its own set of input tiles, denoted

TABLE I
MATHEMATICAL FORMULATION FOR 3-PSDPP

Inputs	$\mathcal{X} = \{1, \dots, X\}, \mathcal{Y} = \{1, \dots, Y\}$, where $X, Y \in \mathbb{N}^*$ $\mathcal{R}_y \subseteq \mathcal{X}, \forall y \in \mathcal{Y}$ $\alpha, \beta \in \mathbb{N}^*$
Outputs	(N, Z, Δ) , where $N, Z, \Delta \in \mathbb{N}^*$ $(p_i)_{i \in \mathcal{N}}$, where $p_i = (d_i, b_i, t_i)$ $(c_j)_{j \in \mathcal{M}}$, where $c_j = (s_j, u_j)$
Constraints	(1) $\forall y \in \mathcal{Y}, \exists j \in \mathcal{M} / s_j = y$ (2) $\forall j \in \mathcal{M}, \forall x \in \mathcal{X}, x \in \mathcal{R}_{s_j} \Rightarrow$ $(\exists a \in \{1, \dots, u_j - \alpha\}, \exists i \in \mathcal{N} / t_i = a, d_i = x \ \&$ $(\forall a' \in \{a + \alpha, \dots, u_j + \beta - 1\}, \forall i' \in \{i + 1, \dots, N\},$ $t_{i'} = a' \Rightarrow b_{i'} \neq b_i))$ (3) $\forall i \in \mathcal{N} \setminus \{1\}, t_i \geq t_{i-1} + \alpha$ (4) $\forall j \in \mathcal{M} \setminus \{1\}, u_j \geq u_{j-1} + \beta$
Objectives	$\min Z, \min N, \min \Delta$

by \mathcal{R}_y . Also, the duration of a prefetch step α , and that of a computation step β , have to be given as input.

Remark 1. *The set of input tiles $(\mathcal{R}_y)_{y \in \mathcal{Y}}$ must be present in the buffers during the whole computation step of the tile y .*

Remark 2. *Each input tile x already prefetched earlier may be reused if it is still present in the buffer.*

2) *Outputs:* a feasible solution to such an instance is defined by $((p_i)_{i \in \mathcal{N}}, (c_j)_{j \in \mathcal{M}}, Z, N, \Delta)$.

- Configuration of the prefetched input tiles:
we denote by $(p_i)_{i \in \mathcal{N}}$ the prefetch sequence, where $p_i = (d_i, b_i, t_i)$ encodes that input tile d_i is prefetched in the buffer b_i at the time t_i .
- Configuration of the computed output tiles:
we denote by $(c_j)_{j \in \mathcal{M}}$ the computation sequence, where $c_j = (s_j, u_j)$ encodes that output tile s_j is to be computed at the time u_j .
- The values for the three criteria (Z, N, Δ) :
we denote by Z the number of buffers; N is the total number of prefetched input tiles; and Δ is the completion time, meaning the total time it takes for the whole operation of the TPU from the beginning of the first prefetch to the end of the last computation of one full image.

3) *Constraints:* the first constraint (1) on solutions is that for each output tile y , there exists a computation step j in which this output tile is computed. The second and main constraint (2) ensures that all the input tiles \mathcal{R}_y required by y have to be prefetched from the external memory to the internal storage area (buffers) before the start date u_j of its associated computation step, and will not be overwritten until its end date. Input tiles already prefetched earlier can be reused, provided they have not been overwritten. Constraints (3) and (4) guarantee that different input (output) tiles cannot be pre-fetched (computed) simultaneously.

4) *Objectives:* in the formulation above, three objectives have to be minimized: the number of prefetches N reflects the main part of the energy consumption; the number of buffers Z of the TPU will account for most of its area, and is related to cost; and the completion time Δ accounts for the performance of the TPU.

C. Sub-Problems of 3-PSDPP

From this multi-objective problem 3-PSDPP, we derive several mono and bi-objective sub-problems. The mono-objective sub-problems we consider are *Minimum Buffers of 3-PSDPP* (MB-PSDPP), in which the number of buffers is to be minimized; *Minimum Prefetches of 3-PSDPP* (MP-PSDPP) in which the number of prefetches is to be minimized; and *Minimum Completion Time of 3-PSDPP* (MCT-PSDPP), in which the completion time is to be minimized.

For the remaining sub-problems to be presented, the number of buffers Z will be fixed as input data. Hence, we consider the *Prefetching and Scheduling Problem* (PSP), where the number of buffers is fixed as input, and the number of prefetches N is to be minimized. In addition, the variant of PSP, where the computation sequence is given as part of the input, is called the *Data Prefetching Problem* (DPP).

We will also consider the bi-objective sub-problem of 3-PSDPP, 2-PSDPP, where the number of buffers Z is fixed, and both N and Δ are to be minimized.

D. Related Work

In the study by Mancini and al. [2], two algorithms for optimizing the running of the TPU produced by the MMOpt tool are proposed.

The first algorithm is M_1 , for which both the number of prefetches N and the completion time Δ are minimized. The second one, called M_2 , aims at minimizing both the number of prefetches N and the number of buffers Z .

The algorithm M_1 proceeds in two steps which are, respectively, *Computations* and *Prefetches*. Furthermore, M_2 comprises three steps, the first two of which are those of M_1 . The third step is *Delay Computations*.

For both algorithms M_1 and M_2 , the authors [2] add a *Destinations* step for determining the sequence of destination buffers.

These different steps are outlined in Fig. 4.

1) *Computations:* this step encodes the order in which a batch of output tiles has to be successively computed, one at a time. The traffic to the external memory is then minimized by optimizing the obtained scheduling.

To construct the computation sequence, the authors [2] solve an instance of the *Asymmetric Traveling Salesman Problem* (ATSP) to find a Hamiltonian path in the complete directed graph \vec{G} whose vertices are the set of output tiles, and whose arcs are weighted by $\varphi(k, l)$, where (k, l) is a pair of output tiles, in which the output tile k will be computed before the output tile l . The function $\varphi(k, l)$ defines the number of additional input tiles to be prefetched for computing the output tile l , when all input tiles shared between k and l are already prefetched.

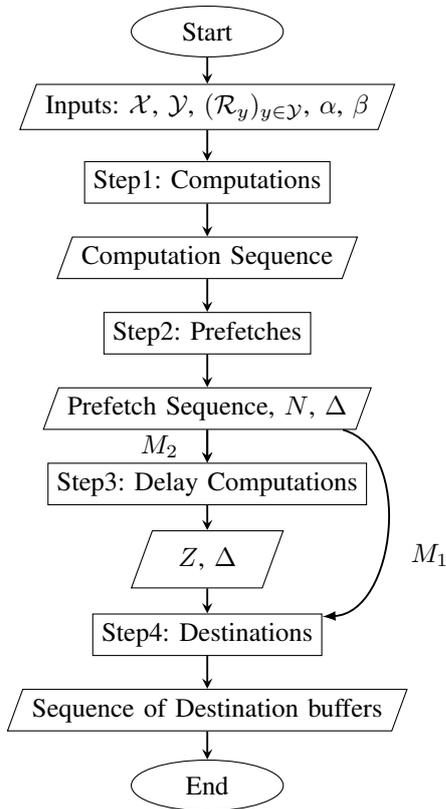


Fig. 4. Flowchart for algorithms M_1 and M_2

2) *Prefetches*: in this step, the authors [2] determine the schedule of prefetches associated to the computation sequence given by step 1. This schedule encodes which input tile should be prefetched from the external memory to the buffers at each moment. In fact, in parallel to each computation step, they prefetch the additional input tiles needed for the next computation.

3) *Delay Computations*: in this step, in order to reduce buffer usage, the authors [2] simply delay some computations by inserting fake computations when necessary. New trade-offs between the embedded memory area and the computing time can be then reached.

4) *Destinations*: this step simply consists in deciding in which buffer to place each prefetched input tile.

To the best of our knowledge, the 3-PSDPP problem has not been studied before in the operations research literature. We now relate some sub-problems of the 3-PSDPP problem to similar problems in the literature. Thus, we focus here on the uniform *Tool Switching Problem* (ToSP) arising in the flexible manufacturing context. The ToSP consists of finding an appropriate job sequence in which jobs will be executed, and an associated sequence of tool switches that minimizes the number of tool loading/unloading operations in the tool magazine of a single tool-switching machine. The general ToSP was first considered by Tang and Denardo [3]. They showed that the ToSP can be solved in polynomial time for a fixed job sequence, *Tooling Problem* (TP), using the *Keep Tools Needed*

Soonest (KTNS) algorithm. On the other hand, when the job sequence is to be determined, Crama and al. showed that the ToSP is already NP-Hard for any fixed tool magazine capacity larger than or equal to 2 [4]. Different optimization techniques, including exact and heuristics methods, have been applied to its resolution (see Bard [5]; Privault and Finke [6]; Laporte and al. [7]; Konak and al. [8]; Amaya and al. [9]; Catanzaro and al. [10]).

IV. MODELS ANALYSIS

For validating the efficiency of the proposed approaches, we develop three lower bounds lb_N , lb_Z , and lb_Δ for the different optimization criteria (N, Z, Δ) . We then give complexity results for some of the mono-objective sub-problems of 3-PSDPP problem described in Section III-C.

A. Lower Bounds

Proposition 1. $X - |\Omega|$ is a lower bound on the number of prefetches for the 3-PSDPP, where Ω denotes the set of input tiles which are not required by any output tile.

Proof: For any solution to some given instance of 3-PSDPP, all input tiles that are required at least once for the computation of an output tile have to be prefetched at least once to some buffer. Hence the total number of prefetches cannot be less than $X - |\Omega|$. ■

Proposition 2. $\max_{y \in \mathcal{Y}} |\mathcal{R}_y|$ is a lower bound on the number of buffers for the 3-PSDPP.

Proof: Fix an instance of 3-PSDPP, and a feasible solution for that instance. When an output tile is computed, all the required input tiles have to be present in the buffers. Hence $\max_{y \in \mathcal{Y}} |\mathcal{R}_y|$ is a lower bound for the number of buffers in the solution. ■

Proposition 3. $lb_1 = \alpha * X' + \beta$ and $lb_2 = \alpha + \beta * Y$ are lower bounds on the completion time Δ for the 3-PSDPP.

Proof: Fix an instance of 3-PSDPP, and a feasible solution for that instance. Since all input tiles have to be loaded before the last computation starts, the completion time is at least $\alpha * X'$ (for the prefetches) plus β (for the computation of the last output tile).

Likewise, all output tiles have to be computed, and no computation can start before a first input tile has been prefetched. Hence the completion time is lower bounded by $\beta * Y$ (computation time for all output tiles) plus α (prefetch time for the first prefetch). ■

Thus, the completion time Δ is lower bounded by the maximum lb_1 and lb_2 ($lb_\Delta = \max\{lb_1, lb_2\}$).

B. Complexity Analysis

To prove that MB-PSDPP can be solved in polynomial time, we give an algorithm for which the number of buffers Z equals its lower bound ($Z_{\min} = lb_Z$). In this algorithm, we first fix the number of buffers Z to $\max_{y \in \mathcal{Y}} |\mathcal{R}_y|$. Then, for each output tile, we prefetch all its required input tiles into the Z buffers before

the corresponding computation step starts. The idea here is that a prefetch step and a computation step are not carried out in parallel.

To prove also that MP-PSDPP is solvable in polynomial time, we give another algorithm for which the number of prefetches N equals its lower bound ($N_{\min} = lb_N$). In this method, we first prefetch successively all the input tiles in \mathcal{X}' , where $\mathcal{X}' = \mathcal{X} \setminus \Omega$ and Ω denotes the set of the input tiles which are not required by any output tile. Then, when the prefetch steps are finished, all output tiles are successively computed.

For the MCT-PSDPP sub-problem, we have not yet been able to determine its complexity.

We now examine the two sub-problems of 3-PSDPP where N is to be minimized, and Z is fixed as input, namely PSP and DPP. We then prove the equivalence between PSP and the tool switching problem ToSP. In the description of the PSP problem, both input and output tiles (\mathcal{X}, \mathcal{Y}) are regarded as ToSP data (tools, jobs). The incidence matrix Tools \times Jobs can then be regarded as the requirements of input tiles needed to compute all the output tiles $(\mathcal{R}_y)_{y \in \mathcal{Y}}$. The fixed number of buffers Z is the analogue of the capacity of the tool magazine. In addition, finding a computation sequence for minimizing the total number of prefetches corresponds to finding a job sequence for minimizing the total number of tools loadings. Thus, PSP is NP-Complete. When the computation sequence is given as input data, the same polynomial reduction works to prove the equivalence of DPP and Tooling Problem (TP). Hence, DPP is polynomially solvable.

This equivalence allows us to adapt the KTNS algorithm, as described by Tang and Denardo for solving the TP [3], to give an optimal solution for DPP. We call this adaptation *KTNS Adapted to DPP* (KAD). On the other hand, in the case of PSP, we developed an algorithm, named *KTNS Adapted to PSP* (KAP), to solve it.

We will also consider the bi-objective problem 2-PSDPP where the number of buffers Z is fixed, and both N and Δ are to be minimized. For solving the 2-PSDPP sub-problem, we developed a new solution approach called *Shifted Prefetches for bi-PSDPP* (SPbP).

To introduce both KAP and SPbP algorithms, respectively, for PSP and 2-PSDPP sub-problems of 3-PSDPP, we now present the specific adaptation KAD that it is an intermediate step in both KAP and SPbP algorithms.

V. SOLUTION METHODS

A. Constructive Heuristic

We propose now a constructive heuristic called H_1 for solving the 3-PSDPP, in which the number of buffers Z equals its number of required input tiles $X - |\Omega|$ and both N and Δ are to be minimized. In this method, the number of prefetches is optimum and we try to get the best possible completion time.

This algorithm proceeds in three phases. For each input tile x , we calculate its number of occurrences $O_c(x)$ in $(\mathcal{R}_y)_{y \in \mathcal{Y}}$. Then, the input tiles are sequenced in their decreasing order of

$O_c(x), \forall x \in \mathcal{X}$. Finally, for each computation is determined when it can be scheduled at the earliest. The corresponding date is the end of the loading of the latest prefetched tile among its required input tiles. Computations are scheduled greedily in this order, while making sure to respect these "at earliest" dates.

B. KTNS Adapted to DPP: KAD

We present an adaptation of the KTNS algorithm for solving the mono-objective sub-problem DPP (described in Section III-C). The KAD adaptation will be the second step of both KAP and SPbP algorithms presented in the following subsections.

We first restate the KTNS policy established by Tang and Denardo [3]. In our case, we can state the KTNS policy in this way:

- 1) At any instant, no input tile is prefetched unless it is required by the next output tile.
- 2) If an input tile must be prefetched, the input tiles that are kept (not removed) are those needed the soonest.

The pseudo-code of the KAD algorithm can be summarized as follows:

Algorithm 1 KAD

Input: $\mathcal{X}, \mathcal{Y}, (\mathcal{R}_y)_{y \in \mathcal{Y}}, (s_j)_{j \in \mathcal{M}}, Z, \alpha, \beta$

Output: $(p_i)_{i \in \mathcal{N}}, N, \Delta$

- 1: $M \leftarrow$ Incidence_Matrix ($\mathcal{X}, \mathcal{Y}, (\mathcal{R}_y)_{y \in \mathcal{Y}}$)
 - 2: $P \leftarrow$ Permute ($M, (s_j)_{j \in \mathcal{M}}$)
 - 3: $P' \leftarrow$ Flip_Blocks (P, Z)
 - 4: $N, (d_i)_{i \in \mathcal{N}} \leftarrow$ Prefetches (P')
 - 5: $(b_i)_{i \in \mathcal{N}} \leftarrow$ Destinations ($((d_i)_{i \in \mathcal{N}}, P, Z)$)
 - 6: $(t_i)_{i \in \mathcal{N}} \leftarrow$ Prefetches_StartDate ($((d_i)_{i \in \mathcal{N}}, \alpha, \beta)$)
 - 7: $(u_j)_{j \in \mathcal{M}} \leftarrow$ Computations_StartDate ($((s_j)_{j \in \mathcal{M}}, (t_i)_{i \in \mathcal{N}}, \alpha, \beta)$)
 - 8: $\Delta \leftarrow$ Completion_Time ($(u_j)_{j \in \mathcal{M}}, \beta$)
-

We now give some explanations about the pseudo-code. Given the computation sequence s_j , a number of buffers Z , and an $X \times Y$ input tile-output tile matrix M , where M_{xy} is 1 if output tile y requires x ($x \in (\mathcal{R}_y)_{y \in \mathcal{Y}}$) and 0 otherwise, we first determine the new incidence matrix P by permuting the columns of M according to the $(s_j)_{j \in \mathcal{M}}$.

In the second step, we determine the set of 0-blocks of P , as shown in Fig. 5. A 0-block is defined as a maximal subset of consecutive zeroes in a row of P . Intuitively, a 0-block is a maximal time interval which input tile i is not needed, but it is needed before and after this interval. Assume now that the 0-blocks of P have been ordered in increasing order of the index of their last column in P , the routine Flip_Blocks (P, Z) flips to 1 as many 0-blocks of P as possible, as long as each column j ($j = 1, \dots, Y$) of P contains no more than Z ones ($Z = 2$ in this example), as shown in Fig. 6.

The resulting matrix is denoted by P' . Then, the routine Prefetches(P') determines the total number of prefetches N , by counting all the blocks of 1 in P' . A 1-block can be defined

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Fig. 5. KTNS-Initial Matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Fig. 6. KTNS-Final Matrix

as a maximal subset of consecutive ones in a row of P' , for which an input tile i is not needed before and after this interval.

To construct the prefetch sequence $(d_i)_{i \in \mathcal{N}}$, we affect, for each output tile (column j in matrix P'), the set of associated input tiles (index of 1-blocks starting in column j) to be prefetched before the corresponding computation step. We determine then the associated sequence of destination buffers $(d_i)_{i \in \mathcal{N}}$, by affecting for each prefetched input tile a free buffer from the Z buffers. After that, we construct the associated schedules of prefetches-computations, in which the input tiles are prefetched one after the other. The same applies for computing the output tiles sequence. In the resulting prefetches-computations schedules, a prefetch step and a computation step are not carried out in parallel. Indeed, each computation step begins on the date where all its required input tiles have been prefetched. The routine `Prefetches_StartDate()` determines the start dates of the prefetches schedules. Similarly, the routine `Computations_StartDate()` determines the start dates of the computations schedules. The routine `Completion_Time()` computes then the completion time Δ .

C. Algorithm KAP for PSP

We have developed the KAP algorithm for solving the PSP sub-problem of 3-PSDPP, in which the number of buffers Z is fixed ($Z \geq \max_{y \in \mathcal{Y}} |\mathcal{R}_y|$) and the number of prefetches N is to be minimized. In contrast to DPP, the order in which the computations have to be carried out is not given as input, and has to be determined.

The KAP algorithm proceeds in two steps as follows:

1) *Step1-Find a Computation Sequence (lines 1–3 of pseudo-code)*: in this step, we determine the computation sequence by solving the same instance of an ATSP as step 1 of both algorithms M_1 and M_2 (see Section III-D) [2].

2) *Step2-KAD Algorithm (line 4 of pseudo-code)*: in second step, we resolve the DPP (described in Section V-B), by the KAD algorithm in order to determine the schedules of both the prefetches and computations with an optimal number of prefetches N .

The pseudo-code of the KAP algorithm can be summarized as follows:

D. Algorithm SPbP for 2-PSDPP

We have developed also the SPbP algorithm for solving the 2-PSDPP sub-problem of 3-PSDPP, in which the number of buffers Z is fixed ($Z \geq \max_{y \in \mathcal{Y}} |\mathcal{R}_y|$) and both the number of prefetches N and the completion time Δ are to be simultaneously minimized.

The SPbP algorithm proceeds in three steps as follows:

Algorithm 2 KAP

Input: $\mathcal{X}, \mathcal{Y}, (\mathcal{R}_y)_{y \in \mathcal{Y}}, Z, \alpha, \beta$

Output: $(p_i)_{i \in \mathcal{N}}, (c_j)_{j \in \mathcal{M}}, N, \Delta$

- 1: Let $\vec{G} = (Y, A)$ be a complete directed graph on Y
 - 2: Let $\varphi: \begin{matrix} A & \rightarrow & \mathbb{N} \\ \vec{k} & \mapsto & |\mathcal{R}_l \setminus \mathcal{R}_k| \end{matrix}$
 - 3: $(s_j)_{j \in \mathcal{M}} \leftarrow \text{Short_Hamiltonian_Cycle}(\vec{G}, \varphi)$
 - 4: $N, (d_i)_{i \in \mathcal{N}}, (b_i)_{i \in \mathcal{N}}, (t_i)_{i \in \mathcal{N}}, (u_j)_{j \in \mathcal{M}}, \Delta \leftarrow \text{KAD}(\mathcal{X}, \mathcal{Y}, (\mathcal{R}_y)_{y \in \mathcal{Y}}, (s_j)_{j \in \mathcal{M}}, Z, \alpha, \beta)$
-

1) *Step1-Find a Computation Sequence (line 1 of pseudo-code)*: this step is the first one of KAP algorithm described in Section V-C.

2) *Step2-KAD Algorithm (lines 2 of pseudo-code)*: this step is the second one of the KAP algorithm described in Section V-C

We give now a detailed description of the third step “Shifting Prefetches”.

3) *Step3-“Shifting Prefetches” (lines 3–9 of pseudo-code)*: In the resulting schedules of prefetches-computations, we apply the idea of “Shifting Prefetches”, in which the completion time Δ is to be minimized. The originality of this step is to allow an overlap between the prefetch and computation steps. In this step, we shift in the prefetches schedule those that can be carried out in parallel with the previous computation step, by checking that no required input tiles of the output tile in this step were overwritten until its end date.

The pseudo-code of the SPbP algorithm can be summarized as follows:

Algorithm 3 SPbP

Input: $\mathcal{X}, \mathcal{Y}, (\mathcal{R}_y)_{y \in \mathcal{Y}}, \alpha, \beta, Z$

Output: $(p_i)_{i \in \mathcal{N}}, (c_j)_{j \in \mathcal{M}}, N, \Delta$

- 1: $(s_j)_{j \in \mathcal{M}} \leftarrow \text{Computation_Sequence}(\mathcal{X}, \mathcal{Y}, (\mathcal{R}_y)_{y \in \mathcal{Y}})$
 - 2: $N, (d_i)_{i \in \mathcal{N}}, (b_i)_{i \in \mathcal{N}}, (t_i)_{i \in \mathcal{N}}, (u_j)_{j \in \mathcal{M}}, \Delta \leftarrow \text{KAD}(\mathcal{X}, \mathcal{Y}, (\mathcal{R}_y)_{y \in \mathcal{Y}}, (s_j)_{j \in \mathcal{M}}, \alpha, \beta, Z)$
 - 3: **for** $j = 2$ **To** \mathcal{M} **do**
 - 4: $\text{Prefetch}[j] \leftarrow \{i \in \mathcal{N} / P'[j][d_i] - P'[j-1][d_i] = 1\}$
 - 5: $\text{Buffer}[j] \leftarrow \{b_i / i \in \text{Prefetch}[j]\}$
 - 6: $\text{Advanced_Prefetch}[j] \leftarrow \{l / l \in \text{Prefetch}[j] \& \text{Buffer}[j][l] \notin b_i(\mathcal{R}_{s_{j-1}})\}$
 - 7: $\text{Not_Advanced_Prefetch}[j] \leftarrow \text{Prefetch}[j] \setminus \text{Advanced_Prefetch}[j]$
 - 8: **end for**
 - 9: $(t_i)_{i \in \mathcal{N}}, (u_j)_{j \in \mathcal{M}}, \Delta \leftarrow \text{Schedule_Prefetches_Computations}((s_j)_{j \in \mathcal{M}}, \text{Advanced_Prefetch}[], \text{Not_Advanced_Prefetch}[], \alpha, \beta)$
-

E. Solutions for 3-PSDPP

Though our 3-PSDPP is a multi-objective optimization problem, we have developed two approaches. The first one

is the KAP algorithm for solving the mono-objective sub-problem PSP, where the number of buffers Z is fixed as input, and the number of prefetches N is to be minimized. The second one is the SPbP algorithm for solving the 2-PSDPP problem, in which the number of buffers Z is fixed, and both N and Δ are to be minimized.

The different steps of both both KAP and SPbP algorithms are outlined in Fig. 7. As shown in this figure, the KAP algorithm represents the two first steps of the SPbP algorithm.

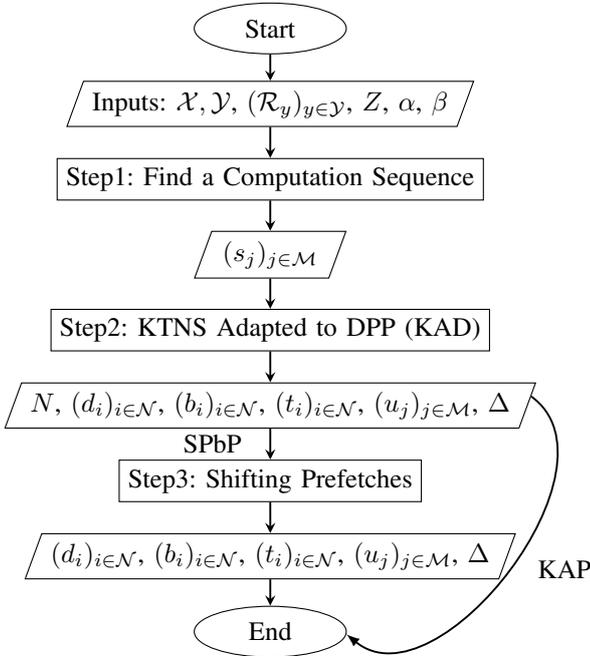


Fig. 7. Flowchart of KAP and SPbP algorithms

Therefore, we can use both KAP and SPbP as two methods for producing solutions to the main multi-objective optimization problem 3-PSDPP. By varying the number of buffers Z , both KAP and SPbP algorithms give a set of different solutions, in order to let the circuit designer pick his favorite compromise solution. It is an asset of these two methods that the circuit designer may choose his solution.

F. Example

In order to illustrate the process of both KAP and SPbP algorithms for solving two sub-problems of 3-PSDPP, where $z = 4$ buffers, let us consider the input data given in Fig. 8 for the case where $x = 5$ input tiles, $y = 3$ output tiles, $\alpha = 2$, and $\beta = 3$ time units.

We first determine the computation sequence $s_j = Y_2, Y_1, Y_3$, by finding a short Hamiltonian cycle in the graph \vec{G} (see Fig. 9).

By running the KAP algorithm on our example, Fig. 10 gives the corresponding prefetches and computations schedules together with values of the outputs.

$$M = \begin{matrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \end{matrix} \begin{pmatrix} Y_1 & Y_2 & Y_3 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Fig. 8. $X \times Y$ Matrix

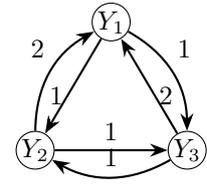


Fig. 9. Graph \vec{G}

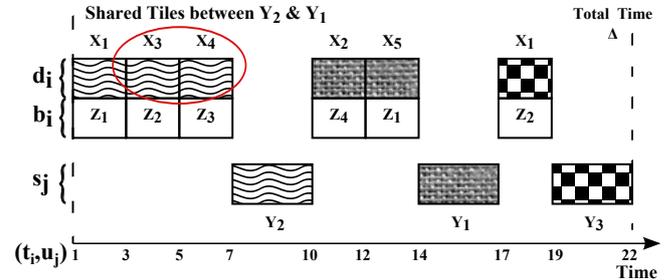


Fig. 10. KAP Schedules

Finally, we apply the “Shifting Prefetches” for minimizing the completion time Δ . The Fig. 11 shows that the completion time Δ is reduced by 2 units of time.

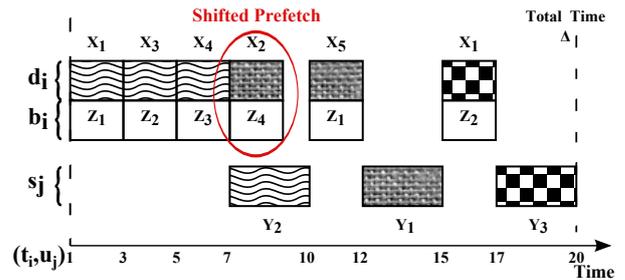


Fig. 11. SPbP Schedules

VI. EXPERIMENTS AND RESULTS

In order to illustrate the practicability and efficiency of both KAP and SPbP algorithms, and the heuristic H_1 , we use a set of 5 benchmarks from real-life non-linear image processing kernels already used by Mancini and al. [1].

A. Data Sets

Table II shows the characteristics of the test instances, together with the values for X (number of input tiles) and Y (number of output tiles to be computed).

As summarized in Table II, the benchmarks are variations of four kernels (fisheye, polar, fd resize, and fd haar) for which the input data structure (multi-resolution (an)isotropic mipmap input data) is modified. In fact, the first three kernels represent geometric non-linear transformations [11], [12]. The fourth kernel creates a pyramidal multi-resolution image [13]. The last one represents a kernel of a face detection application based on haar features [13]. The number of the input image

tiles varies between 350 and 7000 input tiles, and the number of the output tiles varies between 150 and 1200 tiles.

TABLE II
PARAMETERS VALUES OF DATA SETS

N ^o	Kernel	Input data type	X	Y
1	Fisheye	mipmap isotropic	352	158
2	Fisheye	mipmap anisotropic	704	158
3	Polar	mipmap anisotropic	4225	112
4	Fd Resize	mipmap isotropic	1280	1186
5	Fd Haar	pyramidal integral image	7040	428

Table III shows for each kernel the values of different lower bounds ($lb_Z, lb_N, lb_1, lb_2, lb_\Delta$), which are developed in Section IV-A. These lower bounds allow us to evaluate the performances of both proposed approaches KAP and SPbP.

TABLE III
LOWER BOUNDS FOR N, Z , AND Δ

Kernel N ^o	lb_Z	lb_N	lb_1	lb_2	lb_Δ
1	13	224	452	477	477
2	21	360	724	477	724
3	20	244	492	339	492
4	13	429	862	3561	3561
5	96	2272	4548	1287	4548

B. Numerical Results

This section presents an experimental analysis of the performance of both KAP and SPbP algorithms, respectively the heuristic H_1 . The algorithms just described were coded in Python, except the ATSP part which was re-encoded as a TSP, and run through Concorde's Chained Lin-Kernighan implementation [14]. Tests were run on a computer powered by an Intel Core i5 processor with 4 GB of RAM. All our tests were carried out for the case where $\alpha = 2$ and $\beta = 3$ time units.

Table IV summarizes the numerical results for both KAP and SPbP algorithms, where the number of buffers is fixed to Z_1 , respectively to Z_2 , and those of the algorithms M_1 , and M_2 on different data sets described in Table II. For the 5 kernels (given in line 1), the running time of both KAP and SPbP algorithms is in the order of a few minutes. The third line gives the number of prefetches N , the number of buffers Z , and the completion time Δ for the algorithms M_1 , and M_2 (line 2). For both cases Z_1 , and Z_2 (line 3), the N and Δ achieved by the KAP algorithm are then given in line 5. The sixth line (Gain 1) shows the gains as measured relatively to the lower bounds (lb_N and lb_Δ) of KAP algorithm, for both N and Δ relatively to those achieved by algorithms M_1 , and M_2 . Similarly, both N and Δ achieved by the SPbP algorithm are then given in line 7. The eighth line (Gain 2) shows the gains relative to the lower bounds (lb_N and lb_Δ) of SPbP algorithm, for both N and Δ relatively to those achieved by algorithms M_1 , and M_2 . The three last lines give for each

of the algorithms M_1, M_2, KAP , and $SPbP$, the ratio of the achieved completion time Δ to the lower bound lb_Δ (given in Table III). The column Average provides the average gains (%) for all the kernels in the case of Z_1 , respectively, of Z_2 .

As illustrated in Table IV, the fixed Z_1 is larger than its lb_Z , for which the algorithm M_1 reaches the maximum number of buffers and the completion time is minimized. Whereas, Z_2 gives the minimum number of buffers (Z_2 equals its lower bound lb_Z).

By running the KAP algorithm, the traffic to the external memory is reduced with an average reduction of 57.5% (Z_1), respectively, of 36.8% (Z_2). In contrast, the completion time is increased, with an average increase of 14.9% (Z_1), respectively, of 22% (Z_2). This is due to the absence of overlap between the prefetches and computations in the schedules produced by the KAP algorithm.

In addition, due to the reuse of the KAP algorithm as a subroutine of the SPbP algorithm, the traffic to the external memory is reduced with the same average reduction of 57.5% (Z_1), respectively, of 36.8% (Z_2). In contrast to KAP, minimizing Δ by SPbP leads to a 37.1% (Z_1), respectively, to a 25% (Z_2) decrease in average of the completion time.

In the same way, a comparison between the completion time Δ achieved by each of the algorithms M_1, M_2, KAP , and $SPbP$ and the lower bound lb_Δ , is considered. A comparison against the lower bound provides a measure of deviation from optimality. It is used as a performance indicator, and calculated by taking the ratio of Δ to lb_Δ . As shown in Table IV, for both cases Z_1 , and Z_2 , the completion time Δ of SPbP algorithm is in average more closer to the value of lb_Δ than the different values given by each of the algorithms M_1, M_2 , and KAP . It is at most twice the value of lb_Δ . This is explained by inserting "Fake Computation" sometimes to stop processing or prefetching in order to optimize the use of resources. This means also that the SPbP algorithm gives a better completion time Δ than the other methods.

In summary, these numerical experiments show that the schedules produced by both KAP and SPbP algorithms, for a given number of buffers Z , have the optimal number of prefetches N . In contrast to KAP, SPbP gives the best completion time Δ . Hence, these results are numerical evidence validating the efficiency of our proposed approaches as compared to the one currently in use in the MMOpt tool.

In the same way, Table V shows the completion time Δ of the heuristic method H_1 for all the non-linear kernels given in Table II. Thus, a comparison between the completion time Δ obtained by method H_1 and the lower bound lb_Δ , is considered. It is calculated by taking the ratio of Δ to lb_Δ .

TABLE V
COMPARISON BETWEEN Δ OF H_1 AND lb_Δ

Kernel N ^o	1	2	3	4	5
$\Delta (H_1)$	547	736	551	3658	4551
$\Delta (H_1) / lb_\Delta$	1.14	1.01	1.11	1.02	1.00

As shown in Table V, the completion time Δ of H_1 is

TABLE IV
 NUMERICAL RESULTS OF M_1 , M_2 , KAP, AND SPBP

Kernel N°	1		2		3		4		5		Average (Z1) %	Average (Z2) %	
	M_1	M_2	M_1	M_2	M_1	M_2	M_1	M_2	M_1	M_2			
MMOpt	Z	20	13	29	21	28	20	18	13	139	96		
	N	395	395	640	640	478	478	1710	1710	3640	3640		
	Δ	907	976	1341	1457	1021	1081	5075	5129	7899	8070		
	Z_1	Z_2	Z_1	Z_2	Z_1	Z_2	Z_1	Z_2	Z_1	Z_2			
KAP (Fixed Z)	N	298	322	457	517	353	405	1283	1458	2560	2997		
	Δ	1071	1119	1389	1509	1043	1147	6125	6475	6405	7279		
Gain 1	N	56.7	42.6	65.3	43.9	53.4	31.1	33.3	19.6	78.9	47.0	57.5	36.8
	Δ	-38.1	-28.6	-7.7	-7.0	-4.1	-11.2	-69.3	-85.8	44.5	22.4	-14.9	-22.0
SPbP (Fixed Z)	N	298	322	457	517	353	405	1283	1458	2560	2997		
	Δ	795	871	1113	1226	851	947	4629	4916	5849	6789		
Gain 2	N	56.7	42.6	65.3	43.9	53.4	31.1	33.3	19.6	78.9	47.0	57.5	36.8
	Δ	26.0	21.0	36.9	31.5	32.1	22.7	29.4	13.5	61.1	36.3	37.1	25.0
Δ (MMOpt) / lb_{Δ}		1.90	2.04	1.85	2.01	2.07	2.19	1.42	1.44	1.73	1.77	1.79	1.89
Δ (KAP) / lb_{Δ}		2.24	2.34	1.91	2.08	2.11	2.33	1.72	1.81	1.40	1.60	1.87	2.03
Δ (SPbP) / lb_{Δ}		1.66	1.82	1.53	1.69	1.72	1.92	1.29	1.38	1.28	1.49	1.49	1.66

very closer to the value of lb_{Δ} . This means that the lb_{Δ} is a good lower bound on the completion time Δ for the 3-PSDPP problem.

VII. CONCLUSION AND FUTURE WORK

In this paper, we addressed the multi-objective optimization problem 3-PSDPP, that arises in the context of optimizing the memory hierarchy of non-linear kernels in order to enhance some electronic stakes (energy consumption, real time, and cost) in the MMOpt tool. We described some of its mono- and bi-objective variants, proved some lower bounds, and analyzed the complexity of some of them. We then presented a constructive heuristic to solve the 3-PSDPP problem, the KAP algorithm to solve the PSP sub-problem, and the SPbP algorithm to solve the 2-PSDPP sub-problem. The results on the same real-world data set as used by Mancini and al. [1] show a very significant improvement and reduce the amount of transferred data up to 30% and a reduction of the computing time up to 15%.

An interesting area for further research may be the improvement of the proposed methods, and the development of other approaches based on constructive heuristics and exact methods that would provide good solutions for other 3-PSDPP sub-problems. It would also seem interesting to use exact methods such as ILP for solving the NP-Hard variant of the PSP sub-problem. Additional research is also required to determine the complexity of the mono-objective sub-problem MCT-PSDPP, in which the completion time Δ is to be minimized.

REFERENCES

- [1] S. Mancini and F. Rousseau, "Enhancing non-linear kernels by an optimized memory hierarchy in a high level synthesis flow," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2012, pp. 1130–1133.
- [2] —, "Optimisation d'accélérateurs matériels de traitement par incorporation d'un gestionnaire de données et de contrôle dans un flot de HLS," in *Conférence en Parallélisme, Architecture et Système*, 2013.
- [3] C. Tang and E. Denardo, "Models arising from a flexible manufacturing machine, part i: Minimization of the number of tool switches," *Operations Research*, vol. 36, no. 5, pp. 767–777, 1988.
- [4] Y. Crama, A. Kolen, A. Oerlemans, and F. Spieksma, "Minimizing the number of tool switches on a flexible machine," *International Journal of Flexible Manufacturing Systems*, vol. 6, no. 1, pp. 33–54, 1994.
- [5] J. Bard, "A heuristic for minimizing the number of tool switches on a flexible machine," *IIE Transactions*, vol. 20, no. 4, pp. 382–391, 1988. [Online]. Available: <http://dx.doi.org/10.1080/07408178808966195>
- [6] C. Privault and G. Finke, "Modelling a tool switching problem on a single nc-machine," *Journal of Intelligent Manufacturing*, vol. 6, no. 2, pp. 87–94, 1995. [Online]. Available: <http://dx.doi.org/10.1007/BF00123680>
- [7] J. Laporte, J. Salazar-Gonzalez, and F. Semet, "Exact algorithms for the job sequencing and tool switching problem," *IIE Transactions*, vol. 36, no. 1, pp. 37–45, 2004. [Online]. Available: <http://dx.doi.org/10.1080/07408170490257871>
- [8] A. Konak and S. Kulturel-Konak, "An ant colony optimization approach to the minimum tool switching instant problem in flexible manufacturing system," in *2007 IEEE Symposium on Computational Intelligence in Scheduling*, 2007.
- [9] J. Amaya, C. Cotta, and A. Fernández, "A memetic algorithm for the tool switching problem," in *Hybrid metaheuristics*. Springer, 2008, pp. 190–202.
- [10] D. Catanzaro, L. Gouveia, and M. Labbé, "Improved integer linear programming formulations for the job sequencing and tool switching problem," *European Journal of Operational Research*, vol. 244, no. 3, pp. 766–777, 2015.
- [11] A. Thornton and S. Sangwine, "Log-polar sampling incorporating a novel spatially variant filter to improve object recognition," in *Sixth International Conference on Image Processing and Its Applications*, vol. 2, 1997, pp. 776–779.
- [12] N. Bellas, S. Chai, M. Dwyer, and D. Linzmeier, "Real-time fisheye lens distortion correction using automatically generated streaming accelerators," in *17th IEEE Symposium on Field Programmable Custom Computing Machines, FCCM'09.*, 2009, pp. 149–156.
- [13] P. Viola and M. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, no. 2, pp. 137–154, 2004.
- [14] D. Applegate, R. Bixby, W. Cook, and V. Chvátal, *On the solution of traveling salesman problems*. Rheinische Friedrich-Wilhelms-Universität Bonn, 1998.