Embedded vision systems buffer minimization with energy consumption constraint

Khadija Hadj Salem, Tifenn Rault, Alexis Robbes

Embedded vision systems buffer minimization with energy consumption constraint

Khadija HADJ SALEM[1], Tifenn RAULT[1] and Alexis ROBBES[1]

Université de Tours, LIFAT EA 6300, CNRS, ROOT ERL CNRS 7002, 64 avenue Jean Portalis, 37200 Tours
{khadija.hadj-salem, tifenn.rault, alexis.robbes}@univ-tours.fr

Keywords: Embedded vision systems, Scheduling, Tool Switching Problem, Buffers, Linear Integer Programming, Constraint Programming.

1 Introduction

Designing embedded vision systems involves various optimization problems as scheduling image processing. The limited resources of the devices implies to reduce drastically the computation time, the energy consumption and the memory cost. In (Hadj Salem *et. al.* 2018), a image processing scheduling problem is modelized as a variant of the Job Sequencing and tool Switching Problem (SSP). Furthermore, first results was presented about minimizing the makespan and minimizing the number of switches. However, the buffer minimization is only mentioned and remains unstudied. The SSP is a $\mathcal{NP}$-hard problems that arises from computer and manufacturing systems, see (Catanzaro *et. al.* 2015). This problem involves sequencing a finite set of jobs on a single machine and loading restricted subset of tools to a magazine with a limited capacity such that the total number of tool switches is kept to a minimum. In our context, the magazine size is the available memory and the number of switch correspond to the energy consumption.

In this paper, we tackle the buffer minimization as a new extension of the SSP. This dimensioning problem is called *Prefetch-Constrained Minimum Buffers Problem* (P-C-MBP). Our study provides first results, an ILP modelization and a CP approach. We compare these two methods on instances from the literature of SSP and real-world ones. The preliminary results are quite promising and show that the CP is performs better than the ILP.

2 The P-C-MBP Statement

The P-C-MBP involves scheduling a number of output tiles (jobs) on a single machine, under a limited number of prefetches (number of tool switches), such that the resulting number of buffers (magazine slots) is kept to a minimum. This can be formalized as follows: let a P-C-MBP instance be represented by a 4-tuple, $I = (X, Y, \mathcal{R}_y, N)$ where:

- $\mathcal{X} = \{1, \ldots, X\}$ is the set of X input tiles to be prefetched from the external memory to the internal buffers;
- $\mathcal{Y} = \{1, \ldots, Y\}$ is a set of Y independent non-preemptive output tiles (also called tasks) to be computed;
- $(\mathcal{R}_y)_{y \in \mathcal{Y}}$ is the X-dimensional column vector, where $\mathcal{R}_y \subseteq \mathcal{X}$, which defines the set of required input tiles (called prerequisites or tools). These $\mathcal{R}_y$ tiles have to be prefetched from the external memory and must be present in the buffers during the whole corresponding computation step.
  In the same way, we denote by $(\mathcal{R}_x)_{x \in \mathcal{X}}$ the Y-dimensional row vector, where $\mathcal{R}_x \subseteq \mathcal{Y}$, which defines the set of used output tiles for each input tile x.

– N is an integer non-negative number of prefetches, i.e., each input tile can be loaded more than one time. It is assumed that $X < \sum_{y \in \mathcal{Y}} |\mathcal{R}_y|$, otherwise the problem is trivial.

The solution to such an instance is a sequence $y_1, \ldots, y_Y$ determining the order in which the output tiles are computed (executed), and a sequence $(x, z)_1, \ldots, (x, z)_N$ of prefetch configurations determining which input tiles are pretched (loaded) in which buffers (magazine slots) at a certain time. The function objective of P-C-MBP is to minimize the number of buffers, denoted by Z.

We proved that the P-C-BMP problem is $\mathcal{NP}$-hard by showing that if an algorithm exists for solving the P-C-BMP, it can be called iteratively a polynomial number of times to solve the SSP problem. As a consequence, if P-C-BMP was polynomial, so would SSP.

3 A position-based Integer Linear Programming

Due to the relation between the P-C-BMP and the SSP, we can draw inspiration from the existing SSP's ILPs. In fact, (Catanzaro *et. al.* 2015) proposed and compared several ILPs with different categories. In our study, we focused on a position-based ILP, which is very similar to the standard one given by (Tang and Denardo 1987), to which strengthening equalities/inequalities are added.

For all $y \in \mathcal{Y},\ j \in \mathcal{Y}$ and $x \in \mathcal{X}$, we define three sets of binary variables: c_{yj} is equal to 1 if output tile y is computed at position j and 0 otherwise. Similarly, e_{xj} equal to 1 if input tile x exists in buffer at position j (is already loaded) and 0 otherwise. Finally, p_{xj} is equal to 1if input tile x has just been prefetched at position j and 0 if it was already loaded. Then, a valid formulation for the P-C-BMP is the following:

$$\min\ Z$$

Subject to

$$\sum_{j \in \mathcal{Y}} c_{yj} = 1, \forall\ y \in \mathcal{Y} \quad (1)$$

$$\sum_{y \in \mathcal{Y}} c_{yj} = 1, \forall\ j \in \mathcal{Y} \quad (2)$$

$$\sum_{x \in \mathcal{R}_y} e_{xj} \geq |\mathcal{R}_y| * c_{yj}, \forall\ y \in \mathcal{Y},\ j \in \mathcal{Y} \quad (3)$$

$$p_{x1} = e_{x1}, \forall\ x \in \mathcal{X} \quad (4)$$

$$p_{xj} \leq e_{xj}, \forall\ x \in \mathcal{X},\ j \in \mathcal{Y} \backslash \{1\} \quad (5)$$

$$p_{xj} \leq 1 - e_{xj-1}, \forall\ x \in \mathcal{X},\ j \in \mathcal{Y} \backslash \{1\} \quad (6)$$

$$p_{xj} \geq e_{xj} - e_{xj-1}, \forall\ x \in \mathcal{X},\ j \in \mathcal{Y} \backslash \{1\} \quad (7)$$

$$\sum_{x \in \mathcal{X}} \sum_{j \in \mathcal{Y}} p_{xj} \leq N \quad (8)$$

$$\sum_{x \in \mathcal{X}} \sum_{j \in \mathcal{Y}} p_{xj} \geq |\mathcal{X}| \quad (9)$$

$$\sum_{j \in \mathcal{Y}} p_{xj} \geq 1, \forall\ x \in \mathcal{X} \quad (10)$$

$$\sum_{j \in \mathcal{Y}} p_{xj} \leq |\mathcal{R}_x|, \forall\ x \in \mathcal{X} \quad (11)$$

$$Z \geq \sum_{x \in \mathcal{X}} e_{xj}, \forall\ j \in \mathcal{Y} \quad (12)$$

$$Z \geq \max_{y \in \mathcal{Y}} |\mathcal{R}_y|, \forall\ y \in \mathcal{Y} \quad (13)$$

$$Z \leq |\mathcal{X}| \quad (14)$$

$$c_{yj} \in \{0, 1\}, \forall\ y \in \mathcal{Y},\ j \in \mathcal{Y} \quad (15)$$

$$e_{xj}, p_{xj} \in \{0, 1\}, \forall\ x \in \mathcal{X},\ j \in \mathcal{Y} \quad (16)$$

The objective function minimizes the number of buffers defined by Z, under constraints (1) — (16). Constraints (1) — (2) are a set of standard assignment constraints. Constraints (3) are the requirement constraints which define the relation between prefetches and computations steps. Similarly, constraints (4) — (7) define the relation between the loading and prefetching of input tiles. Constraint sets (8) — (9) and (10) — (11) give bounds on

prefetches number and on prefetches number for each input tile $x, \forall x \in \mathcal{X}$, respectively. In the same way, constraints (12) compute a lower bound of Z in relation to the prefetches number for each computation position $j, \forall j \in \mathcal{Y}$, while constraints (13) — (14) give an upper/lower bounds on buffers number. Finally, constraints (15) — (16) set the ranges of the variables.

To strengthen the ILP given before, we provide two valid inequalities (17) — (18), in which constraint (17) ensures that the first computation must takes place during the first half of the possible position, while constraints (18) gives a lower bound on the number of times in which input tiles must exist in the buffer.

$$\sum_{j=1}^{|\mathcal{Y}|/2} c_{1j} = 1 \qquad (17) \qquad\qquad \sum_{x \in \mathcal{X}} \sum_{j \in \mathcal{Y}} e_{xj} \geq \sum_{y \in \mathcal{Y}} |\mathcal{R}_y| \qquad (18)$$

4 A Constraint Programming Approach

To the best of our knowledge, it appears that constraint programming paradigm (CP) has been little considered in the SSP's literature. Thus, we introduce here a new CP model for the P-C-MBP using concepts present in IBM ILOG CP Optimizer.

We first define two sets of variables as follows:

- I_y: the interval variable for each output tile (task) $y \in \mathcal{Y}$.
- $I_{x,j}$: the interval variable for each input tile (prerequisite) $x \in \mathcal{X}$, and for each interval number $j \in \{1, \ldots, j_{\max}\}$, where $j_{\max} = N - |\mathcal{X}| + 1$. If $j = 1$ this variable is mandatory, while for $j > 0$ the interval is optional. This means that an input tile x is loaded at least once, and possibly more.

We also use the following *Cumul functions*:

- $\texttt{nbXScheduled} = \sum_{x \in \mathcal{X}, j \in \{1,\ldots,j_{\max}\}} \texttt{StepAtStart}(I_{x,j}, 1)$;
- $\texttt{nbOverlap} = \sum_{x \in \mathcal{X}, j \in \{1,\ldots,j_{\max}\}} \texttt{Pulse}(I_{x,j}, 1)$;
- $\texttt{necessaryTiles}\,[y] = \sum_{x \in \mathcal{R}_y, j \in \{1,\ldots,j_{\max}\}} \texttt{Pulse}(I_{x,j}, 1), \forall y \in \mathcal{Y}$.

The objective is to minimize the buffers number Z, subject to the following constraints:

$$\texttt{noOverlap}(\texttt{I}_\texttt{y}) \quad \forall\, y \in \mathcal{Y} \qquad (19)$$

$$\texttt{nbOverlap}(\texttt{I}_{\texttt{x,j}}) \quad \forall\, x \in \mathcal{X}, \forall j \in \{1, .., j_{\max}\} \qquad (20)$$

$$\texttt{nbXScheduled} \leq N \qquad (21)$$

$$\texttt{nbOverlap} \leq Z \qquad (22)$$

$$\texttt{alwaysIn}(\texttt{necessaryTiles}[\texttt{y}], \texttt{I}_\texttt{y}, |\mathcal{R}_\texttt{y}|, |\mathcal{R}_\texttt{y}|) \quad \forall y \in \mathcal{Y} \qquad (23)$$

5 Computational Experiments

To evaluate the different approaches, we present first numerical results. All experiments were performed on a Intel Core i5 processor, 2.67 GHz machine, equipped with 4 GB of RAM and operating system Windows. The ILP is solved by CPLEX and the CP approach is implemented using IBM ILOG CP Optimizer. The CPU time limit for each run on each

problem instance is 300 seconds. The number of prefetches N is set to $|\mathcal{X}|$, which is its smallest possible value.

Experiments were made using two kind of data-sets possessing different characteristics. We first considered a collection of 16 data-sets for the well-known SSP, downloadable at http://www.unet.edu.ve/∼jedgar/ToSP/ToSP.htm. We then considered a set of 10 benchmarks from real-life non-linear image processing kernels for MMOpt software already used in (Mancini and Rousseau 2012) and (Hadj Salem *et. al.* 2018). These instances are reduced by using a dominance property, which removes each output tile that needs a subset of input tiles, which is already used at least once by another output tile.

In our preliminary tests on SSP data-sets (instances goes from size 9×10 to 40×60), both ILP and CP models were able to optimally solve within the time limit only instances with (X, Y) under $(15, 10)$ input/output tiles. In contrast, for the other instances, they give similar results, except for few ones for which the CP is a bit better.
In the case of bigger instances of MMOpt (greater than 64×64 input/output tiles), Table 1 gives the detailed numerical results of ILP and CP models. In this table, the **Gap(%)** is computed as follow: $100 * (Z_{\text{ILP}} - Z_{\text{CP}})/Z_{\text{CP}}$.

Table 1. Numerical results for ILP & CP using MMOpt instances

Instance	X	Y	$Z_{\textbf{ILP}}$	**time$_{\textbf{ILP}}$**	$Z_{\textbf{CP}}$	**time$_{\textbf{CP}}$**	**Gap(%)**
test_2D_0	256	64	**4**	101.39	**4**	0.09	**0**
test_2D_1	64	256	**1**	2,5	**1**	0,11	**0**
test_polaire_0	146	90	146	300	68	300	114.7
test_polaire_1	80	60	80	–	46	–	73.91
test_polaire_2	244	82	244	–	142	–	71.83
test_fisheye_0	176	103	176	–	109	–	61.46
test_fisheye_1	224	103	224	–	139	–	61.15
test_fisheye_2	360	103	360	–	230	–	56.52
test_fd_0	429	300	429	–	349	–	22.92
test_fd_1	2272	206	2272	–	2081	–	9.17

As illustrated in Table 1, we can see that both ILP and CP models can solve only the first two instances. For the rest, the CP model is definitely better than ILP and provide relatively good upper bounds.

Overall, ILP model fails to find good bounds, specifically in the case of MMOpt instances. In the other hand, CP model seems to be able to handle well this kind of problem. Nonetheless, extensive tests, some improvements and comparison with other mat-heuristics should be considered.

References

Catanzaro D., L. Gouveia and M. Labbé, 2015, "Improved integer linear programming formulations for the job Sequencing and tool Switching Problem", *European Journal of Operational Research*, Vol. 244, pp. 766–777.

Hadj Salem K., Y. Kieffer and M. Mancini, 2018, "Meeting the Challenges of Optimized Memory Management in Embedded Vision Systems Using Operations Research", *Recent Advances in Computational Optimization: Results of the Workshop on Computational Optimization WCO 2016*, pp. 177–205.

Mancini M., F. Rousseau, 2012, "Enhancing non-linear kernels by an opti-mized memory hierarchy in a high level synthesis flow", *Conference on Design, Automation and Test in Europe*, pp. 1130–1133.

Tang C.S., E.V. Denardo, 1987, "Models arising from a flexible manufacturing machine, Part I: Minimization of the number of tool switches", *Operations Research*, pp. 767–777.