Bi-criteria Pipeline Mappings for Parallel Image Processing

Anne Benoit[1], Harald Kosch[2], Veronika Rehn-Sonigo[1], and Yves Robert[1]

[1] LIP, ENS Lyon, 46 Allée d'Italie, 69364 Lyon Cedex 07, France
{Anne.Benoit,Veronika.Sonigo,Yves.Robert}@ens-lyon.fr
[2] University of Passau, Innstr. 43, 94032 Passau, Germany
Harald.Kosch@uni-passau.de

Abstract. Mapping workflow applications onto parallel platforms is a challenging problem, even for simple application patterns such as pipeline graphs. Several antagonistic criteria should be optimized, such as throughput/period and latency (or a combination). Typical applications include digital image processing, where images are processed in steady-state mode. In this paper, we study the bi-criteria mapping (minimizing period and latency) of the JPEG encoding on a cluster of workstations. We present an integer linear programming formulation for this NP-hard problem, and we present an in-depth performance evaluation of several polynomial heuristics.

Keywords: pipeline, workflow application, multi-criteria, optimization, JPEG encoding.

1 Introduction

This work considers the problem of mapping workflow applications onto parallel platforms. This is a challenging problem, even for simple application patterns. For homogeneous architectures, several scheduling and load-balancing techniques have been developed but the extension to heterogeneous clusters makes the problem more difficult.

Structured programming approaches rule out many of the problems which the low-level parallel application developer is usually confronted to, such as deadlocks or process starvation. We therefore focus on pipeline applications, as they can easily be expressed as algorithmic skeletons. More precisely, in this paper, we study the mapping of a particular pipeline application: we focus on the JPEG encoder (baseline process, basic mode). This image processing application transforms numerical pictures from any format into a standardized format called JPEG. This standard was developed almost 20 years ago to create a portable format for the compression of still images and new versions are created until now (see http://www.jpeg.org/). Meanwhile, several parallel algorithms have been proposed [9]. JPEG (and later JPEG 2000) is used for encoding still images in Motion-JPEG (later MJ2). These standards are commonly employed in IP-cams and are part of many video applications in the world of game consoles. Motion-JPEG (M-JPEG) has been adopted and further developed to several other formats, e.g., AMV (alternatively known as MTV) which is a proprietary video file format designed to be consumed on low-resource devices. The manner of encoding in M-JPEG and subsequent formats leads to a flow of still image coding, hence pipeline mapping is appropriate.

We consider the different steps of the encoder as a linear pipeline of stages, where each stage gets some input, has to perform several computations and transfers the output to the next stage. The corresponding mapping problem can be stated informally as follows: which stage to assign to which processor? We require the mapping to be interval-based, i.e., a processor is assigned an interval of consecutive stages. Two key optimization parameters emerge. On the one hand, we target a high throughput, or short period, in order to be able to handle as many images as possible per time unit. On the other hand, we aim at a short response time, or latency, for the processing of each image. These two criteria are antagonistic: intuitively, we obtain a high throughput with many processors to share the work, while we get a small latency by mapping many stages to the same processor in order to avoid the cost of inter-stage communications.

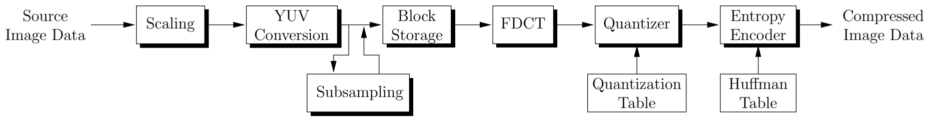


Fig. 1. Steps of the JPEG encoding

2 Framework

Principles of JPEG encoding. Here we briefly present the mode of operation of a JPEG encoder (see [14] for further details). The encoder consists in seven pipeline stages, as shown in Fig. 1. In the first stage, the image is scaled to have a multiple of an 8x8 pixel matrix, and the standard even claims a multiple of 16x16. In the next stage a color space conversion is performed from the RGB to the YUV-color model. The sub-sampling stage is an optional stage, which, depending on the sampling rate, reduces the data volume: as the human eye can dissolve luminosity more easily than color, the chrominance components are sampled more rarely than the luminance components. Admittedly, this leads to a loss of data. The last preparation step consists in the creation and storage of so-called MCUs (Minimum Coded Units), which correspond to 8x8 pixel blocks in the picture. The next stage is the core of the encoder. It performs a Fast Discrete Cosine Transformation (FDCT) (eg. [15]) on the 8x8 pixel blocks which are interpreted as a discrete signal of 64 values. After the transformation, every point in the matrix is represented as a linear combination of the 64 points. The quantizer reduces the image information to the important parts. Depending on the quantization factor and quantization matrix, irrelevant frequencies are reduced. Thereby quantization errors can occur, that are remarkable as quantization noise or block generation in the encoded image. The last stage is the entropy encoder, which performs a modified Huffman coding.

Applicative framework. On the theoretical point of view, we consider a pipeline of n stages $\mathcal{S}_k$, $1 \leq k \leq n$. Tasks are fed into the pipeline and processed from stage to stage, until they exit the pipeline after the last stage. The k-th stage $\mathcal{S}_k$ first receives an input from the previous stage, of size δ_{k-1}, then performs a number of w_k computations, and finally outputs data of size δ_k to the next stage. These three operations are performed sequentially. The first stage $\mathcal{S}_1$ receives an input of size δ_0 from the outside world, while the last stage $\mathcal{S}_n$ returns the result, of size δ_n, to the outside world, thus these particular stages behave in the same way as the others.

On the practical point of view, we consider the applicative pipeline of the JPEG encoder as presented in Fig. 1 and its seven stages.

Target platform. We target a platform with p processors P_u, $1 \leq u \leq p$, fully interconnected as a (virtual) clique. There is a bidirectional link $\mathsf{link}_{u,v} : P_u \rightarrow P_v$ between any processor pair P_u and P_v, of bandwidth $\mathsf{b}_{u,v}$. The speed of processor P_u is denoted as s_u, and it takes X/s_u time-units for P_u to execute X floating point operations. We enforce a linear cost model for communications: it takes X/b time-units to send (resp. receive) a message of size X to (resp. from) P_v. Communications contention is taken care of by enforcing the *one-port* model [3].

Bi-criteria interval mapping problem. We seek to map intervals of consecutive stages onto processors [13]. Intuitively, assigning several consecutive tasks to the same processor will increase their computational load, but may well dramatically decrease communication requirements. We search for a partition of $[1..n]$ into $m \leq p$ intervals $I_j = [d_j, e_j]$ such that $d_j \leq e_j$ for $1 \leq j \leq m$, $d_1 = 1$, $d_{j+1} = e_j + 1$ for $1 \leq j \leq m - 1$ and $e_m = n$.

The optimization problem is to determine the best mapping, over all possible partitions into intervals, and over all processor assignments. The objective can be to minimize either the period, or the latency, or a combination: given a threshold period, what is the minimum latency that can be achieved? and the counterpart: given a threshold latency, what is the minimum period that can be achieved?

The decision problem associated to this bi-criteria interval mapping optimization problem is NP-hard, since the period minimization problem is NP-hard for interval-based mappings (see [2]).

3 Linear Program Formulation

We present here an integer linear program to compute the optimal interval-based bi-criteria mapping on *Fully Heterogeneous* platforms, respecting either a fixed latency or a fixed period. We consider a framework of n stages and p processors, plus two fictitious extra stages $\mathcal{S}_0$ and $\mathcal{S}_{n+1}$ respectively assigned to P_{in} and P_{out}. First we need to define a few variables.

For $k \in [0..n+1]$ and $u \in [1..p] \cup \{\mathsf{in}, \mathsf{out}\}$, $x_{k,u}$ is a boolean variable equal to 1 if stage $\mathcal{S}_k$ is assigned to processor P_u; we let $x_{0,\mathsf{in}} = x_{n+1,\mathsf{out}} = 1$, and $x_{k,\mathsf{in}} = x_{k,\mathsf{out}} = 0$ for $1 \leq k \leq n$. For $k \in [0..n]$, $u, v \in [1..p] \cup \{\mathsf{in}, \mathsf{out}\}$ with

$u \neq v$, $z_{k,u,v}$ is a boolean variable equal to 1 if stage $\mathcal{S}_k$ is assigned to P_u and stage S_{k+1} is assigned to P_v: hence $\mathsf{link}_{u,v} : P_u \rightarrow P_v$ is used for the communication between these two stages. If $k \neq 0$ then $z_{k,\mathsf{in},v} = 0$ for all $v \neq \mathsf{in}$ and if $k \neq n$ then $z_{k,u,\mathsf{out}} = 0$ for all $u \neq \mathsf{out}$. For $k \in [0..n]$ and $u \in [1..p] \cup \{\mathsf{in}, \mathsf{out}\}$, $y_{k,u}$ is a boolean variable equal to 1 if stages $\mathcal{S}_k$ and S_{k+1} are both assigned to P_u; we let $y_{k,\mathsf{in}} = y_{k,\mathsf{out}} = 0$ for all k, and $y_{0,u} = y_{n,u} = 0$ for all u. For $u \in [1..p]$, $\mathsf{first}(u)$ is an integer variable which denotes the first stage assigned to P_u; similarly, $\mathsf{last}(u)$ denotes the last stage assigned to P_u. Thus P_u is assigned the interval $[\mathsf{first}(u), \mathsf{last}(u)]$. Of course $1 \leq \mathsf{first}(u) \leq \mathsf{last}(u) \leq n$. T_{opt} is the variable to optimize, so depending on the objective function it corresponds either to the period or to the latency.

We list below the constraints that need to be enforced. For simplicity, we write $\sum_u$ instead of $\sum_{u \in [1..p] \cup \{\mathsf{in}, \mathsf{out}\}}$ when summing over all processors. First there are constraints for processor and link usage: every stage is assigned a processor, i.e., $\forall k \in [0..n+1]$, $\sum_u x_{k,u} = 1$. Every communication either is assigned a link or collapses because both stages are assigned to the same processor: $\forall k \in [0..n]$, $\sum_{u \neq v} z_{k,u,v} + \sum_u y_{k,u} = 1$. If stage $\mathcal{S}_k$ is assigned to P_u and stage $\mathcal{S}_{k+1}$ to P_v, then $\mathsf{link}_{u,v} : P_u \rightarrow P_v$ is used for this communication: $\forall k \in [0..n], \forall u, v \in [1..p] \cup \{\mathsf{in}, \mathsf{out}\}, u \neq v,\ x_{k,u} + x_{k+1,v} \leq 1 + z_{k,u,v}$. If both stages $\mathcal{S}_k$ and $\mathcal{S}_{k+1}$ are assigned to P_u, then $y_{k,u} = 1$: $\forall k \in [0..n], \forall u \in [1..p] \cup \{\mathsf{in}, \mathsf{out}\},\ x_{k,u} + x_{k+1,u} \leq 1 + y_{k,u}$.

If stage $\mathcal{S}_k$ is assigned to P_u, then necessarily $\mathsf{first}_u \leq k \leq \mathsf{last}_u$. We write this constraint as: $\forall k \in [1..n], \forall u \in [1..p],\ \mathsf{first}_u \leq k.x_{k,u} + n.(1 - x_{k,u})$ and $\forall k \in [1..n], \forall u \in [1..p],\ \mathsf{last}_u \geq k.x_{k,u}$. Furthermore, if stage $\mathcal{S}_k$ is assigned to P_u and stage $\mathcal{S}_{k+1}$ is assigned to $P_v \neq P_u$ (*i.e.*, $z_{k,u,v} = 1$) then necessarily $\mathsf{last}_u \leq k$ and $\mathsf{first}_v \geq k+1$ since we consider intervals. We write this constraint as: $\forall k \in [1..n-1], \forall u, v \in [1..p], u \neq v,\ \mathsf{last}_u \leq k.z_{k,u,v} + n.(1 - z_{k,u,v})$ and $\forall k \in [1..n-1], \forall u, v \in [1..p], u \neq v,\ \mathsf{first}_v \geq (k+1).z_{k,u,v}$.

The latency of the schedule is bounded by T_{latency}:

$$\sum_{u=1}^{p} \sum_{k=1}^{n} \left[\left(\sum_{t \neq u} \frac{\delta_{k-1}}{\mathsf{b}_{t,u}} z_{k-1,t,u} \right) + \frac{\mathsf{w}_k}{\mathsf{s}_u} x_{k,u} \right] + \left(\sum_{u \in [1..p] \cup \{\mathsf{in}\}} \frac{\delta_n}{\mathsf{b}_{u,\mathsf{out}}} z_{n,u,\mathsf{out}} \right) \leq T_{\mathsf{latency}}$$

and $t \in [1..p] \cup \{\mathsf{in}, \mathsf{out}\}$.

There remains to express the period of each processor and to constrain it by T_{period}: $\forall u \in [1..p]$,

$$\sum_{k=1}^{n} \left\{ \left(\sum_{t \neq u} \frac{\delta_{k-1}}{\mathsf{b}_{t,u}} z_{k-1,t,u} \right) + \frac{\mathsf{w}_k}{\mathsf{s}_u} x_{k,u} + \left(\sum_{v \neq u} \frac{\delta_k}{\mathsf{b}_{u,v}} z_{k,u,v} \right) \right\} \leq T_{\mathsf{period}}.$$

Finally, the objective function is either to minimize the period T_{period} respecting the fixed latency T_{latency} or to minimize the latency T_{latency} with a fixed period T_{period}. So in the first case we fix T_{latency} and set $T_{\mathsf{opt}} = T_{\mathsf{period}}$. In the second case T_{period} is fixed a priori and $T_{\mathsf{opt}} = T_{\mathsf{latency}}$. With this mechanism the objective function reduces to minimizing T_{opt} in both cases.

4 Overview of the Heuristics

The problem of bi-criteria interval mapping of workflow applications is NP-hard [2], so in this section we briefly describe polynomial heuristics to solve it. See [2] for a more complete description or refer to the Web at:

`http://graal.ens-lyon.fr/~vsonigo/code/multicriteria/`

In the following, we denote by n the number of stages, and by p the number of processors. We distinguish two sets of heuristics. The heuristics of the first set aim to minimize the latency respecting an a priori fixed period. The heuristics of the second set minimize the counterpart: the latency is fixed a priori and we try to achieve a minimum period while respecting the latency constraint.

4.1 Minimizing Latency for a Fixed Period

All the following heuristics sort processors by non-increasing speed, and start by assigning all the stages to the first (fastest) processor in the list. This processor becomes *used*.

H1-Sp-mono-P: Splitting mono-criterion. At each step, we select the used processor j with the largest period and we try to split its stage interval, giving some stages to the next fastest processor j' in the list (not yet used). This can be done by splitting the interval at any place, and either placing the first part of the interval on j and the remainder on j', or the other way round. The solution which minimizes $max(period(j), period(j'))$ is chosen if it is better than the original solution. Splitting is performed as long as we have not reached the fixed period or until we cannot improve the period anymore.

H2-Sp-bi-P: Splitting bi-criteria. This heuristic uses a binary search over the latency. For this purpose at each iteration we fix an authorized increase of the optimal latency (which is obtained by mapping all stages on the fastest processor), and we test if we get a feasible solution via splitting. The splitting mechanism itself is quite similar to **H1-Sp-mono-P** except that we choose the solution that minimizes $max_{i\in\{j,j'\}}(\frac{\Delta latency}{\Delta period(j)})$ within the authorized latency increase to decide where to split. While we get a feasible solution, we reduce the authorized latency increase for the next iteration of the binary search, thereby aiming at minimizing the mapping global latency.

H3-3-Sp-mono-P: 3-splitting mono-criterion. At each step we select the used processor j with the largest period and we split its interval into three parts. For this purpose we try to map two parts of the interval on the next pair of fastest processors in the list, j' and j'', and to keep the third part on processor j. Testing all possible permutations and all possible positions where to cut, we choose the solution that minimizes $max(period(j), period(j'), period(j''))$.

H4-3-Sp-bi-P: 3-splitting bi-criteria. In this heuristic the choice of where to split is more elaborated: it depends not only of the period improvement, but also of the latency increase. Using the same splitting mechanism as in **H3-3-Sp-mono-P**, we select the solution that minimizes $max_{i\in\{j,j',j''\}}(\frac{\Delta latency}{\Delta period(i)})$.

Here $\Delta latency$ denotes the difference between the global latency of the solution before the split and after the split. In the same manner $\Delta period(i)$ defines the difference between the period before the split (achieved by processor j) and the new period of processor i.

4.2 Minimizing Period for a Fixed Latency

As in the heuristics described above, first of all we sort processors according to their speed and map all stages on the fastest processor.

H5-Sp-mono-L: Splitting mono-criterion. This heuristic uses the same method as **H1-Sp-mono-P** with a different break condition. Here splitting is performed as long as we do not exceed the fixed latency, still choosing the solution that minimizes $max(period(j), period(j'))$.

H6-Sp-bi-L: Splitting bi-criteria. This variant of the splitting heuristic works similarly to **H5-Sp-mono-L**, but at each step it chooses the solution which minimizes $max_{i\in\{j,j'\}}(\frac{\Delta latency}{\Delta period(i)})$ while the fixed latency is not exceeded.

Remark. In the context of M-JPEG coding, minimizing the latency for a fixed period corresponds to a fixed coding rate, and we want to minimize the response time. The counterpart (minimizing the period respecting a fixed latency L) corresponds to the question: if I accept to wait L time units for a given image, which coding rate can I achieve? We evaluate the behavior of the heuristics with respect to these questions in Section 5.

5 Experiments and Simulations

In the following experiments, we study the mapping of the JPEG application onto clusters of workstations.

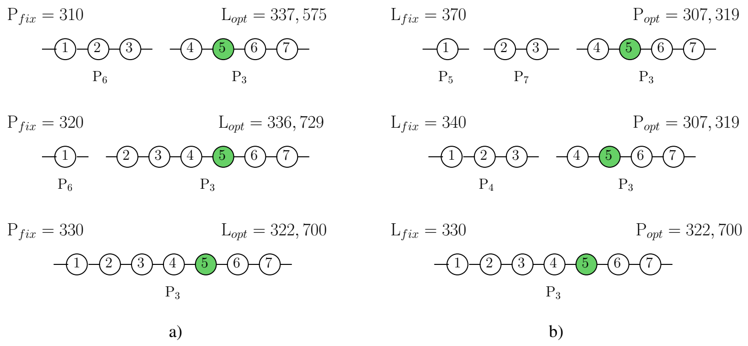
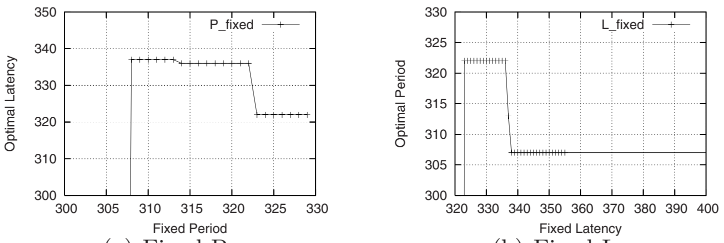


Fig. 2. LP solutions strongly depend on fixed initial parameters

Influence of fixed parameters. In this first test series, we examine the influence of fixed parameters on the solution of the linear program. As shown in

Fig. 2, the division into intervals is highly dependant of the chosen fixed value. The optimal solution to minimize the latency (without any supplemental constraints) obviously consists in mapping the whole application pipeline onto the fastest processor. As expected, if the period fixed in the linear program is not smaller than the latter optimal mono-criterion latency, this solution is chosen. Decreasing the value for the fixed period imposes to split the stages among several processors, until no more solution can be found. Fig. 2(a) shows the division into intervals for a fixed period. A fixed period of $T_{\mathsf{period}} = 330$ is sufficiently high for the whole pipeline to be mapped onto the fastest processor, whereas smaller periods lead to splitting into intervals. We would like to mention, that for a period fixed to 300, there exists no solution anymore. The counterpart - fixed latency - can be found in Fig. 2(b). Note that the first two solutions find the same period, but for a different latency. The first solution has a high value for latency, which allows more splits, hence larger communication costs. Comparing the last lines of Fig. 2(a) and (b), we state that both solutions are the same, and we have $T_{\mathsf{period}} = T_{\mathsf{latency}}$. Finally, expanding the range of the fixed values, a sort of bucket behavior becomes apparent: Increasing the fixed parameter has in a first time no influence, the LP still finds the same solution until the increase crosses an unknown bound and the LP can find a better solution. This phenomenon is shown in Fig. 3.

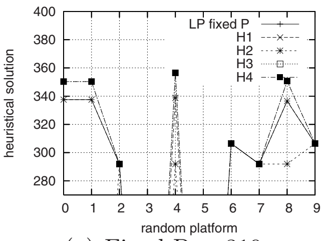


(a) Fixed P. (b) Fixed L.

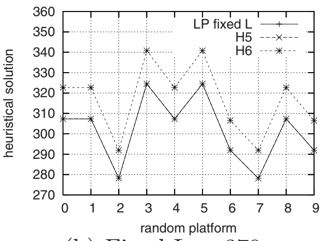
Fig. 3. Bucket behavior of LP solutions

Assessing heuristic performance. The comparison of the solution returned by the LP program, in terms of optimal latency respecting a fixed period (or the converse) with the heuristics is shown in Fig. 4. The implementation is fed with the parameters of the JPEG encoding pipeline and computes the mapping on 10 randomly created platforms with 10 processors. On platforms 3 and 5, no valid solution can be found for the fixed period. There are two important points to mention. First, the solutions found by H2 often are not valid, since they do not respect the fixed period, but they have the best ratio latency/period. Fig. 5(b) plots some more details: H2 achieves good latency results, but the fixed period of P=310 is often violated. This is a consequence of the fact that the fixed period value is very close to the feasible period. When the tolerance for the period is bigger, this heuristic succeeds to find low-latency solutions. Second, all solutions, LP and heuristics, always keep the stages 4 to 7 together (see Fig. 2

for an example). As stage 5 (DCT) is the most costly in terms of computation, the interval containing these stages is responsible for the period of the whole application. Finally, in the comparative study H1 always finds the optimal period for a fixed latency and we therefore recommend this heuristic for period optimization. In the case of latency minimization for a fixed period, then H5 is to use, as it always finds the LP solution in the experiments. This is a striking result, especially given the fact that the LP integer program may require a long time to compute the solution (up to 11389 seconds in our experiments), while the heuristics always complete in less than a second, and find the corresponding optimal solution.

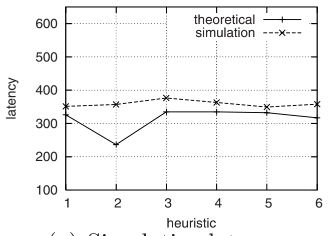


(a) Fixed P = 310.

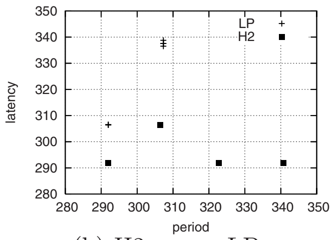


(b) Fixed L = 370.

Fig. 4. Behavior of the heuristics (comparing to LP solution)



(a) Simulative latency.



(b) H2 versus LP.

Fig. 5. MPI simulation results

MPI simulations on a cluster. This last experiment performs a JPEG encoding simulation. All simulations are made on a cluster of homogeneous Optiplex GX 745 machines with an Intel Core 2 Duo 6300 of 1,83Ghz. Heterogeneity is enforced by increasing and decreasing the number of operations a processor has to execute. The same holds for bandwidth capacities. For simplicity we use a MPI program whose stages have the same communication and computation parameters as the JPEG encoder, but we do not encode real images (hence the name simulation, although we use an actual implementation with MPICH).

In this experiment the same random platforms with 10 processors and fixed parameters as in the theoretical experiments are used. We measured the latency of the simulation, even for the heuristics of fixed latency, and computed the average over all random platforms. Fig. 5(a) compares the average of the theoretical results of the heuristics to the average simulative performance. The simulative behavior nicely mirrors the theoretical behavior, with the exception of H2 (see Fig. 5(b)). Here once again, some solutions of this heuristic are not valid, as they do not respect the fixed period.

6 Related Work

The blockwise independent processing of the JPEG encoder allows to apply simple data parallelism for efficient parallelization. Many papers have addressed this fine-grain parallelization opportunity [5,12]. In addition, parallelization of almost all stages, from color space conversion, over DCT to the Huffman encoding has been addressed [1,7]. Recently, with respect to the JPEG2000 codec, efficient parallelization of wavelet coding has been introduced [8]. All these works target the best speed-up with respect to different architectures and possible varying load situations. Optimizing the period and the latency is an important issue when encoding a pipeline of multiple images, as for instance for Motion JPEG (M-JPEG). To meet these issues, one has to solve in addition to the above mentioned work a bi-criteria optimization problem, i.e., optimize the latency, as well as the period. The application of coarse grain parallelism seems to be a promising solution. We propose to use an interval-based mapping strategy allowing multiple stages to be mapped to one processor which allows meeting the most flexible the domain constraints (even for very large pictures). Several pipelined versions of the JPEG encoding have been considered. They rely mainly on pixel or blockwise parallelization [6,10]. For instance, Ferretti et al. [6] uses three pipelines to carry out concurrently the encoding on independent pixels extracted from the serial stream of incoming data. The pixel and block-based approach is however useful for small pictures only. Recently, Sheel et al. [11] consider a pipeline architecture where each stage presents a step in the JPEG encoding. The targeted architecture consists of Xtensa LX processors which run subprograms of the JPEG encoder program. Each program accepts data via the queues of the processor, performs the necessary computation, and finally pushes it to the output queue into the next stage of the pipeline. The basic assumptions are similar to our work, however no optimization problem is considered and only runtime (latency) measurements are available. The schedule is static and set according to basic assumptions about the image processing, e.g., that the DCT is the most complex operation in runtime.

7 Conclusion

In this paper, we have studied the bi-criteria (minimizing latency and period) mapping of pipeline workflow applications, from both a theoretical and practical point of view. On the theoretical side, we have presented an integer linear programming formulation for this NP-hard problem. On the practical side, we have studied in depth the interval mapping of the JPEG encoding pipeline on a cluster of workstations. Owing to the LP solution, we were able to characterize a bucket behavior in the optimal solution, depending on the initial parameters. Furthermore, we have compared the behavior of some polynomial heuristics to the LP solution and we were able to recommended two heuristics with almost optimal behavior for parallel JPEG encoding. Finally, we evaluated the heuristics running a parallel pipeline application with the same parameters as a JPEG encoder. The heuristics were designed for general pipeline applications, and some of them were aiming at applications with a large number of stages (3-splitting), thus a priori not very efficient on the JPEG encoder. Still, some of these heuristics reach the optimal solution in our experiments, which is a striking result.

A natural extension of this work would be to consider further image processing applications with more pipeline stages or a slightly more complicated pipeline architecture. Naturally, our work extends to JPEG 2000 encoding which offers among others wavelet coding and more complex multiple-component image encoding [4]. Another extension is for the MPEG coding family which uses lagged feedback: the coding of some types of frames depends on other frames. Differentiating the types of coding algorithms, a pipeline architecture seems again to be a promising solution architecture.

References

1. Agostini, L.V., Silva, I.S., Bampi, S.: Parallel color space converters for JPEG image compression. Microelectronics Reliability 44, 697 (2004)
2. Benoit, A., Rehn-Sonigo, V., Robert, Y.: Multi-criteria Scheduling of Pipeline Workflows. In: HeteroPar 2007, Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks. IEEE Computer Society Press, Los Alamitos (2007)
3. Bhat, P., Raghavendra, C., Prasanna, V.: Efficient collective communication in distributed heterogeneous systems. Journal of Parallel and Distributed Computing 63, 251 (2003)
4. Christopoulos, C., Skodras, A., Ebrahimi, T.: The JPEG2000 still image coding system: an overview. IEEE Trans. on Consumer Electronics 46, 1103 (2000)
5. Falkemeier, J., Joubert, G.: Parallel image compression with JPEG for multimedisa applications. High Performance Computing: Technologies, Methods and Applications, Advances in Parallel Computing, 379–394 (1995)
6. Ferretti, M., Boffadossi, M.: A Parallel Pipelined Implementation of LOCO-I for JPEG-LS. In: 17th International Conference on Pattern Recognition (ICPR 2004), vol. 1, pp. 769–772 (2004)
7. Kumaki, T., et al.: Acceleration of DCT Processing with Massive-Parallel Memory-Embedded SIMD Matrix Processor. IEICE Trans. on Information and Systems - LETTER- Image Processing and Video Processing E90-D, 1312 (2007)
8. Meerwald, P., Norcen, R., Uhl, A.: Parallel JPEG2000 Image Coding on Multiprocessors. In: IPDPS 2002. IEEE Computer Society Press, Los Alamitos (2002)
9. Monnes, P., Furht, B.: Parallel JPEG Algorithms for Still Image Processing. In: Southeastcon 1994. Creative Technology Transfer - A Global Affair. Proceedings of the 1994 IEEE, pp. 375–379 (1994)

10. Papadonikolakis, M., Pantazis, V., Kakarountas, A.P.: Efficient high-performance ASIC implementation of JPEG-LS encoder. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE2007). IEEE Communications Society Press (2007)
11. Shee, S.L., Erdos, A., Parameswaran, S.: Architectural Exploration of Heterogeneous Multiprocessor Systems for JPEG. International Journal of Parallel Programming 35 (2007)
12. Shen, K., Cook, G., Jamieson, L., Delp, E.: An overview of parallel processing approaches to image and video compression. In: Image and Video Compression, Proc. SPIE, vol. 2186, pp. 197–208 (1994)
13. Subhlok, J., Vondran, G.: Optimal latency-throughput tradeoffs for data parallel pipelines. In: ACM SPAA 1996, pp. 62–71. ACM Press, New York (1996)
14. Wallace, G.K.: The JPEG still picture compression standard. Commun. ACM 34, 30 (1991)
15. Wen-Hsiung, C., Smith, C., Fralick, S.: A Fast Computational Algorithm for the Discrete Cosine Tranfsorm. IEEE Trans. on Communications 25, 1004 (1977)