Multi-criteria Scheduling of Pipeline Workflows (and Application to the JPEG Encoder)

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. 33, 94032 Passau, Germany*
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 antagonist criteria should be optimized, such as throughput and latency (or a combination). A typical application class is digital image coding, where images are processed in steady-state mode. In this paper, we study the general bi-criteria mapping problem (minimizing period and latency) for pipeline graphs on communication homogeneous platforms. We present an integer linear programming formulation for this NP-hard problem. Furthermore, we provide several efficient polynomial bi-criteria heuristics, whose relative performance is evaluated through extensive simulations. As a case-study, we provide simulations and MPI experimental results for the JPEG encoder application pipeline on a cluster of workstations.

I. Introduction

Mapping applications onto parallel platforms is a difficult challenge. Several scheduling and load-balancing techniques have been developed for homogeneous architectures (see [3] for a survey) but the advent of heterogeneous clusters has rendered the mapping problem even more difficult. Typically, such clusters are composed of different-speed processors interconnected either by plain Ethernet (the low-end version) or by a high-speed switch (the high-end counterpart), and they constitute the experimental platform of choice in most academic or industry research departments.

In this context of heterogeneous platforms, a structured programming approach rules out many of the problems which the low-level parallel application developer is usually confronted with, such as deadlocks or process starvation. Moreover, many real applications draw from a range of well-known solution paradigms, such as pipelined or farmed computations. High-level approaches based on algorithmic skeletons ([4], [5]) identify such patterns and seek to make it easy for an application developer to tailor such a paradigm to a specific problem. A library of skeletons is provided to the programmer, who can rely on these already coded patterns to express the communication scheme within his/her own application. Moreover, the use of a particular skeleton carries with it considerable information about implied scheduling dependencies, which we believe can help address the complex problem of mapping a distributed application onto a heterogeneous platform. In this paper, we study the mapping of a particular pipeline application: we focus on the JPEG encoder application pipeline (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 [6]. 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. Each stage has its own communication

A fraction of this work appears in HeteroPar 2007 [1] and in ICSS 2008 [2].

and computation requirements: it reads an input file from the previous stage, processes the data and outputs a result to the next stage. For each data set, initial data is input to the first stage, and final results are output from the last stage. The pipeline workflow operates in synchronous mode: after some latency due to the initialization delay, a new task is completed every period. The period is defined as the longest cycle-time to operate a stage.

Key metrics for a given workflow are the throughput and the latency. The throughput measures the aggregate rate of processing of data, and it is the rate at which data sets can enter the system. Equivalently, the inverse of the throughput, defined as the period, is the time interval required between the beginning of the execution of two consecutive data sets. The latency is the time elapsed between the beginning and the end of the execution of a given data set, hence it measures the response time of the system to process the data set entirely. Note that it may well be the case that different data sets have different latencies (because they are mapped onto different processor sets), hence the latency is defined as the maximum response time over all data sets. Minimizing the latency is antagonistic to minimizing the period, and trade-offs should be found between these criteria. In this paper, we focus on bi-criteria approaches, i.e., minimizing the latency under period constraints, or the converse.

The problem of mapping pipeline skeletons onto parallel platforms has received some attention, and we survey related work in Section VIII. Each pipeline stage can be seen as a sequential procedure which may perform disc accesses or write data in the memory for each task. This data may be reused from one task to another, and thus the rule of the game is always to process the tasks in a sequential order within a stage. Moreover, due to the possible local memory accesses, a given stage must be mapped onto a single processor: we cannot process half of the tasks on a processor and the remaining tasks on another without exchanging intra-stage information, which might be costly and difficult to implement. In other words, a processor that is assigned a stage will execute the operations required by this stage (input, computation and output) for all the tasks fed into the pipeline.

The optimization 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. We target *Communication Homogeneous* platforms, with identical links but different speed processors, which introduce a first degree of heterogeneity. Such platforms correspond to networks of workstations interconnected by a LAN, which constitute the typical experimental platforms in most academic or research departments. Because this bi-criteria optimization problem is NP-hard (see Section IV), the main objective of this paper is to derive efficient polynomial bi-criteria heuristics, and to assess their performances through simulations and MPI experiments.

The rest of the paper is organized as follows. We briefly present the mode of operation of a JPEG encoder in Section II. Section III is devoted to the presentation of the target optimization problems. Next in Section IV we proceed to the complexity results. A linear program formulation can be found in Section V. In Section VI, we introduce several polynomial heuristics to solve the mapping problem. These heuristics are compared through simulations, whose results are analyzed in Section VII. Section VIII gives an overview of related work. Finally, we state some concluding remarks in Section IX.

II. Principles of JPEG Encoding

Here we briefly present the mode of operation of a JPEG encoder (see [7] 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 distinguish 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) (e.g., [8]) 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.

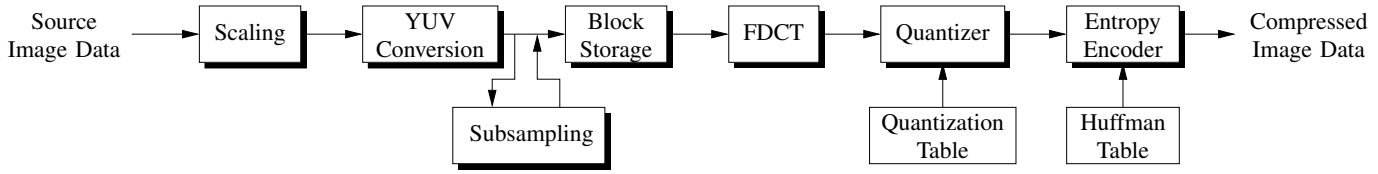


Fig. 1. Steps of the JPEG encoding.

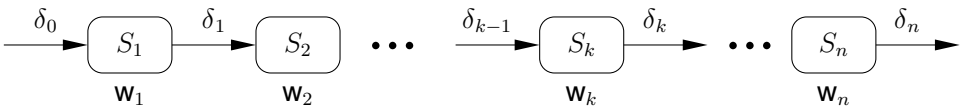


Fig. 2. The application pipeline.

III. Framework

Applicative framework. From the theoretical point of view, we consider a pipeline of n stages $\mathcal{S}_k$, $1 \leq k \leq n$, as illustrated on Fig. 2. 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 $\mathcal{S}_k$ receives an input from the previous stage, of size δ_{k-1}, performs a number of w_k computations, and 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.

From 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 $\mathsf{b}_{u,v}$. Note that we do not need to have a physical link between any processor pair. Instead, we may have a switch, or even a path composed of several physical links, to interconnect P_u and P_v; in the latter case we would retain the bandwidth of the slowest link in the path for the value of $\mathsf{b}_{u,v}$. In the most general case, we have fully heterogeneous platforms, with different processor speeds and link capacities, but we restrict ourselves in this paper to *Communication Homogeneous* platforms with different-speed processors ($\mathsf{s}_u \neq \mathsf{s}_v$) interconnected by links of same capacities ($\mathsf{b}_{u,v} = \mathsf{b}$). They correspond to networks of different-speed processors or workstations interconnected by either plain Ethernet or by a high-speed switch, and they constitute the typical experimental platforms in most academic or industry research departments.

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 also enforce a linear cost model for communications, hence 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 [9], [10]. In this model, a given processor can be involved in a single communication at any time-step, either a send or a receive. However, independent communications between distinct processor pairs can take place simultaneously. The one-port model seems to fit the performance of some current MPI implementations, which serialize asynchronous MPI sends as soon as message sizes exceed a few megabytes [11].

Bi-criteria mapping problem. The general mapping problem consists in assigning application stages to platform processors. For the sake of simplicity, we can assume that each stage $\mathcal{S}_k$ of the application pipeline is mapped onto a distinct processor (which is possible only if $n \leq p$). However, such one-to-one mappings may be unduly restrictive, and a natural extension is to search for interval mappings, i.e., allocation functions where each participating processor is assigned an interval of consecutive stages. Intuitively, assigning several consecutive tasks to the same processors will increase their computational load, but may well dramatically decrease communication requirements. In fact, the best interval mapping may turn out to be a one-to-one mapping, or instead may enroll only a very small number of fast computing processors interconnected by high-speed links.

Interval mappings constitute a natural and useful generalization of one-to-one mappings (not to speak of situations where $p < n$, where interval mappings are mandatory), and such mappings have been studied by Subhlock et al. [12], [13]. The cost model associated to interval mappings is the following. 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 $e_m = n$. Interval I_j is mapped onto processor $P_{\mathsf{alloc}(j)}$, and the period is expressed as

$$T_{\mathsf{period}} = \max_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j - 1}}{\mathsf{b}} + \frac{\sum_{i=d_j}^{e_j} \mathsf{w}_i}{\mathsf{s}_{\mathsf{alloc}(j)}} + \frac{\delta_{e_j}}{\mathsf{b}} \right\} \tag{1}$$

The latency is obtained by the following expression (data sets traverse all stages, and only interprocessor communications need be paid for):

$$T_{\mathsf{latency}} = \sum_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j - 1}}{\mathsf{b}} + \frac{\sum_{i=d_j}^{e_j} \mathsf{w}_i}{\mathsf{s}_{\mathsf{alloc}(j)}} \right\} + \frac{\delta_n}{\mathsf{b}} \tag{2}$$

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?

IV. Complexity Results

In this section we study the complexity of the bi-criteria optimization problem for an interval-based mapping of pipeline applications onto *Communication Homogeneous* platforms. On such platforms, minimizing the latency is not difficult, while minimizing the period is NP-hard. Note that for *Fully Heterogeneous* platforms, minimizing the latency also becomes NP-hard (see [14]).

Proposition 1: The optimal interval-based mapping of pipeline applications which minimizes the latency can be determined in polynomial time.

Proof: The minimum latency can be achieved by mapping the whole interval onto the fastest processor j, resulting in the latency $\left(\sum_{i=1}^{n} \mathsf{w}_i\right) / \mathsf{s}_j$. If a slower processor is involved in the mapping, the latency increases, following equation (2), since part of the computations will take longer, and communications may occur. ■

Theorem 1: The period minimization problem is NP-complete.

Proof: The proof is available in [15]. Interestingly, this result is a consequence of the fact that the natural extension of the chains-to-chains problem [16] to different-speed processors is NP-hard. ■

Proposition 2: All bi-criteria problems are NP-hard.

Proof: This is a direct consequence of Theorem 1. ■

V. Linear Program Formulation

We present here an integer linear program to compute the optimal interval-based bi-criteria mapping on fully heterogeneous platforms (even more general than our target *Communication Homogeneous* 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 $u \in [1..p]$, first_u is an integer variable which denotes the first stage assigned to P_u; similarly, 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.

VI. HEURISTICS

In this section several polynomial heuristics for *Communication Homogeneous* platforms are presented. We restrict to such platforms because, as already pointed out in Section III, clusters made of different-speed processors interconnected by either plain Ethernet or a high-speed switch constitute the typical experimental platforms in most academic or industry research departments. Moreover, this bi-criteria problem is already NP-hard because of the period minimization problem. Note that it would be much more difficult to design efficient heuristics for *Fully Heterogeneous* platforms since the latency minimization problem also becomes NP-hard. In fact, it would become hard to predict the latency of a mapping before the mapping is entirely known.

In the following, we denote by n the number of stages, and by p the number of processors. Note that the code for all these heuristics can be found on the Web (see [17]).

A. Minimizing Latency for a Fixed Period

In the first set of heuristics, the period is fixed a priori, and we aim at minimizing the latency while respecting the prescribed 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*.

- **P1 2-Split mono: 2-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.

- **P2 2-Split bi: 2-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 **P1 2-Split mono** except that we choose the solution that minimizes $\max_{i\in\{j,j'\}}(\frac{\Delta latency}{\Delta period(i)})$ 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 global latency of the mapping.
- **P3 3-Split mono: 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 $\max(period(j), period(j'), period(j''))$.
- **P4 3-Split bi: 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 **P3 3-Split mono**, 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.

B. Minimizing Period for a Fixed Latency

In this second set of heuristics, latency is fixed, and we try to achieve a minimum period while respecting the latency constraint. As in the heuristics described above, first of all we sort processors according to their speed and map all stages on the fastest processor. The approach used here is the converse of the heuristics where we fix the period, as we start with an optimal solution concerning latency. Indeed, at each step we downgrade the solution with respect to its latency but improve it regarding its period.

- **L1 2-Split mono: 2-Splitting mono-criterion –** This heuristic uses the same method as **P1 2-Split mono**, 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 $\max(period(j), period(j'))$.
- **L2 2-Split bi: 2-Splitting bi-criteria –** This variant of the splitting heuristic works similarly to **L1 2-Split mono**, 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.

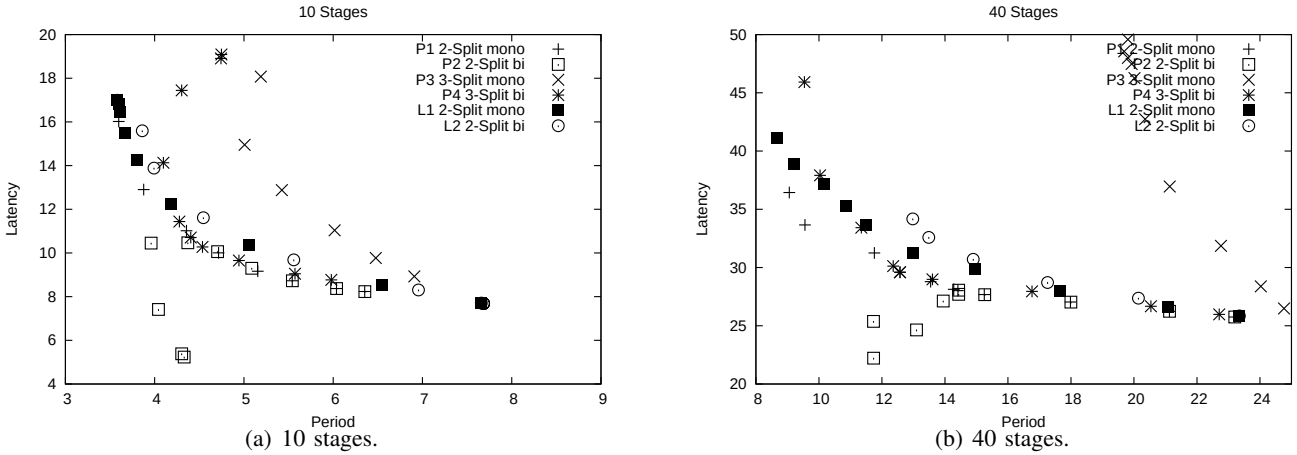
VII. Experimental Results

In this section we present a performance evaluation of our polynomial heuristics. Section VII-A evaluates the heuristics in a general context, i.e., using randomly generated workflow applications. Then Section VII-B is dedicated to the evaluation of our heuristics for a real world application, namely the JPEG encoder application pipeline.

A. General Experiments

Several general experiments have been conducted in order to assess the performance of the heuristics described in Section VI. First we describe the experimental setting, then we report the results, and finally we provide a summary.

1) Experimental Setting: We have generated a set of random applications with $n \in \{5, 10, 20, 40\}$ stages and a set of random *Communication Homogeneous* platforms with $p = 10$ or $p = 100$ processors. In all the experiments, we fix $\mathsf{b} = 10$ for the link bandwidths. Moreover, the speed of each processor is randomly chosen as an integer between 1 and 20. We keep the latter range of variation throughout the experiments, while we vary the range of the application parameters from one set of simulations to the other. Indeed, although there are four categories of parameters to play with, i.e., the values of δ, w, s and b, we can see from equations (1) and (2) that only the relative ratios $\frac{\delta}{\mathsf{b}}$ and $\frac{\mathsf{w}}{\mathsf{s}}$ have an impact on the performance. Each experimental value reported in the following has been calculated as an average over 50 randomly chosen application/platform pairs. For each of these pairs, we report the performance of the six heuristics described in Section VI.



(a) 10 stages. (b) 40 stages.

Fig. 3. $p = 10$ - (E1) Balanced communications/computations, and homogeneous communications.

We report four sets of experiments conducted both for $p = 10$ and $p = 100$ processors. For each experiment, we vary some key application/platform parameter to assess the impact of this parameter on the performance of the heuristics. The first two experiments deal with applications where communications and computations have the same order of magnitude, and we study the impact of the degree of heterogeneity of the communications, i.e., of the variation range of the δ parameter:

- **(E1): balanced communication/computation, and homogeneous communications.** In the first set of experiments, the application communications are homogeneous, we fix $\delta_i = 10$ for $i = 0..n$. The computation time required by each stage is randomly chosen between 1 and 20. Thus, communications and computations are balanced within the application.
- **(E2): balanced communications/computations, and heterogeneous communications.** In the second set of experiments, the application communications are heterogeneous, chosen randomly between 1 and 100. Similarly to Experiment 1, the computation time required by each stage is randomly chosen between 1 and 20. Thus, communications and computations are still relatively balanced within the application.

The last two experiments deal with imbalanced applications: the third experiment assumes large computations (large value of the w to δ ratio), and the fourth one reports results for small ones (small value of the w to δ ratio):

- **(E3): large computations.** In this experiment, the applications are much more demanding on computations than on communications, making communications negligible with respect to computation requirements. We choose the communication time between 1 and 20, while the computation time of each application is chosen between 10 and 1000.
- **(E4): small computations.** The last experiment is the opposite to Experiment 3 since computations are now negligible compared to communications. The communication time is still chosen between 1 and 20, but the computation time is now chosen between 0.01 and 10.

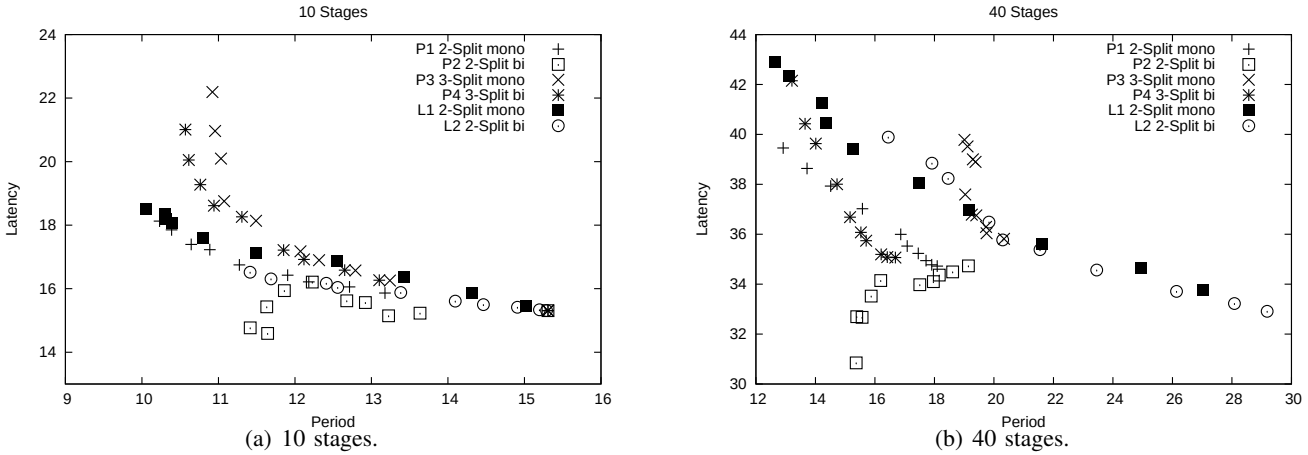
2) Results: Results for the entire set of experiments can be found on the Web (see [17]). In the following we only present the most significant plots.

For each heuristic with a fixed period, we compute for a set of periods the resulting latency, and we plot the latency as a function of the period. If a small period is chosen, the heuristic may fail to find a feasible solution. In this case, there is no corresponding point in the plot and the largest value of such period is reported in the failure thresholds table (see Table I).

Similarly, the values of resulting period obtained for the heuristics with a fixed latency are plotted, reporting the period as a function of the latency. In this case, the heuristic may fail to find a solution if the fixed latency is too small, and latency thresholds are detailed in Table I.

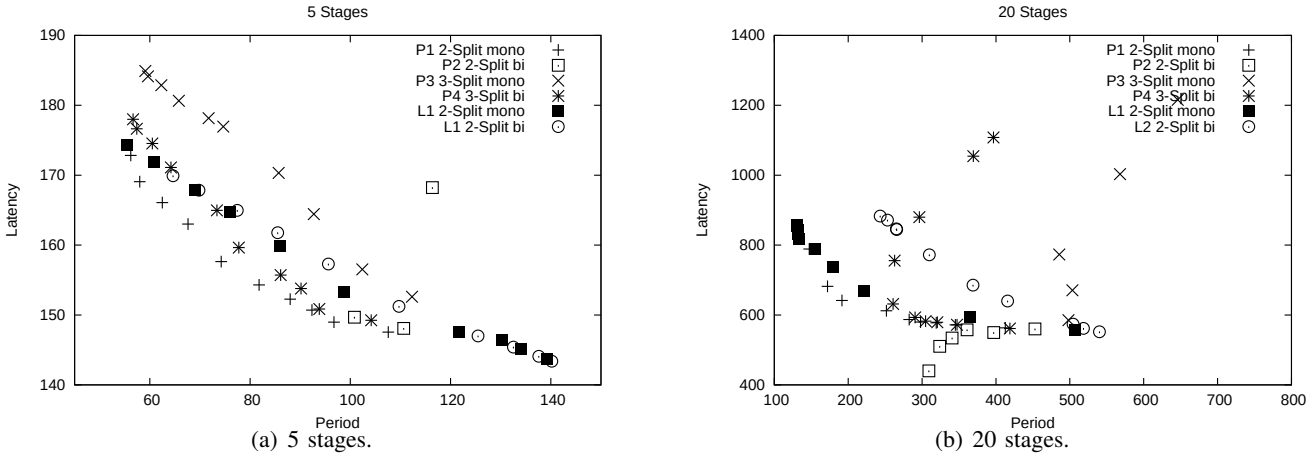
This way, both categories of heuristics can be plotted into the same figure for a given experimental setup, and all heuristics can be compared one to each other.

a) With $p = 10$ processors: For experiment (E1) we see that all heuristics follow the same curve shape, with the exception of heuristic **P2 2-Split bi** (see Figure 3), which has a different behavior. We observe this general



(a) 10 stages. (b) 40 stages.

Fig. 4. $p = 10$ - (E2) Balanced communications/computations, and heterogeneous communications.



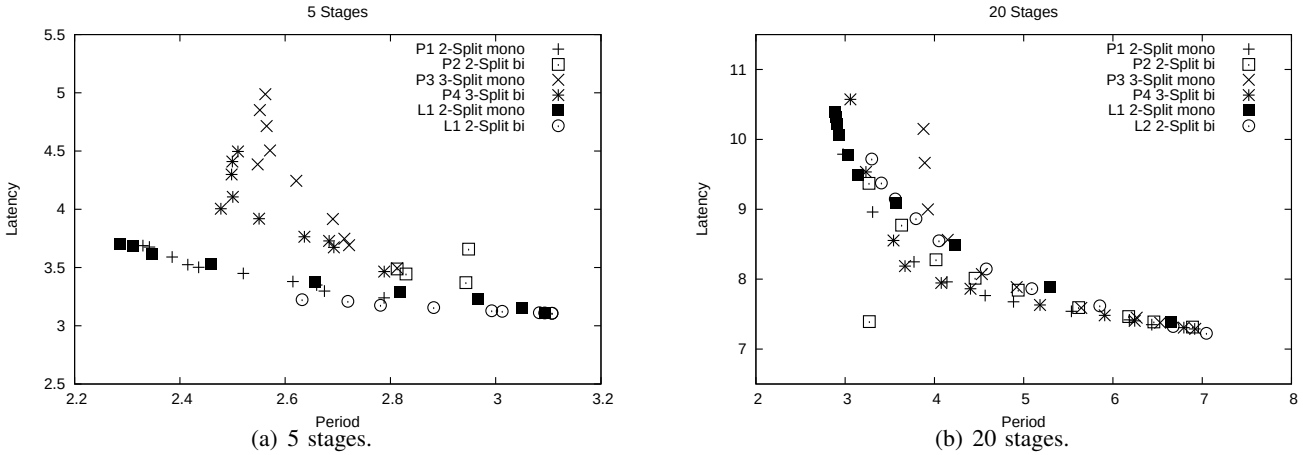
(a) 5 stages. (b) 20 stages.

Fig. 5. $p = 10$ - (E3) Large computations.

behavior of the different heuristics in all the experiments. The heuristic **P2 2-Split bi** initially finds a solution with relatively small period and latency, and then tends to increase both. The other five heuristics achieve small period times at the price of long latencies and then seem to converge to a somewhat shorter latency. We observe that the simplest splitting heuristics perform very well: **P1 2-Split mono** and **L1 2-Split mono** achieve the best period, and **P1 2-Split mono** has the lower latency. **P2 2-Split bi** minimizes the latency with competitive period sizes. Its counterpart **L2 2-Split bi** performs poorly in comparison. **P3 3-Split mono** and **L2 2-Split bi** cannot keep up with the other heuristics (but the latter achieves better results than the former). In the middle range of period values, **P4 3-Split bi** achieves comparable latency values with those of **P1 2-Split mono** and **P2 2-Split bi**.

For experiment (E2) (Cf. Figure 4), if we leave aside **P2 2-Split bi**, we see that **P1 2-Split mono** outperforms the other heuristics almost everywhere with the following exception: with 40 stages and a large fixed period, **P4 3-Split bi** obtains the better results. **P2 2-Split bi** achieves by far the best latency times, but the period times are not as good as those of **P1 2-Split mono** and **P4 3-Split bi**. We observe that the competitiveness of **P4 3-Split bi** increases with the increase of the number of stages. **L1 2-Split mono** achieves period values just as small as **P1 2-Split mono** but the corresponding latency is higher and once again it performs better than its bi-criteria counterpart **L2 2-Split bi**. The poorest results are obtained by **P3 3-Split mono**.

The results of experiment (E3) are much more scattered than in the other experiments (E1, E2 and E4) and this difference even increases with rising n (see Figure 5). When $n = 5$, the results of the different heuristics are almost parallel so that we can state the following hierarchy: **P1 2-Split mono**, **P4 3-Split bi**, **L1 2-Split mono**, **L2 2-Split bi** and finally **P3 3-Split mono**. For this experiment, **P2 2-Split bi** achieves rather poor results. With the increase of the number of stages n, the performance of **P2 2-Split bi** gets better and this heuristic achieves the



(a) 5 stages. (b) 20 stages.

Fig. 6. $p = 10$ - (E4) Small computations.

TABLE I
FAILURE THRESHOLDS WITH $p = 10$.

Exp.	Heur.	Number of stages				Exp.	Heur.	Number of stages			
		5	10	20	40			5	10	20	40
E1	P1	3.0	3.3	5.0	5.0	E3	P1	50.0	70.0	100.0	250.0
	P2	3.0	4.7	9.0	18.0		P2	50.0	140.0	450.0	950.0
	P3	3.0	4.0	5.0	5.0		P3	50.0	90.0	250.0	400.0
	P4	3.3	3.3	6.0	10.0		P4	100.0	140.0	300.0	650.0
	L1	4.5	6.0	13.0	25.0		L1	140.0	270.0	500.0	1000.0
	L2	4.5	6.0	13.0	25.0		L2	140.0	270.0	500.0	1000.0
E2	P1	9.7	10.0	11.0	11.0	E4	P1	2.2	2.3	2.3	2.3
	P2	10.3	10.0	12.0	19.0		P2	2.4	2.7	3.7	7.0
	P3	10.0	10.0	11.0	11.0		P3	2.4	2.7	3.0	4.0
	P4	11.3	11.0	13.0	15.0		P4	2.8	2.7	3.0	4.0
	L1	11.7	15.0	22.0	32.0		L1	3.0	4.0	7.0	11.0
	L2	11.7	15.0	22.0	32.0		L2	3.0	4.0	7.0	11.0

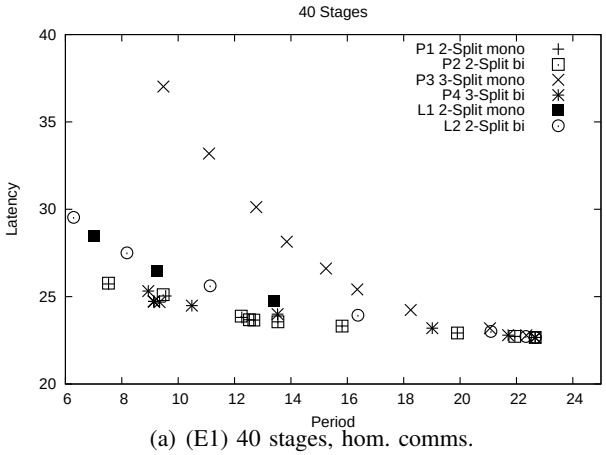
best latency, but its period values cannot compete with **P1 2-Split mono** and **P4 3-Split bi**. These latter heuristics achieve very good results concerning period durations. On the contrary, **P3 3-Split mono** exploses its period and latency times. **P4 3-Split bi** loses its second position for small period times compared to **L1 2-Split mono**, but when period times are higher it recovers its position in the hierarchy.

In experiment (E4), **P3 3-Split mono** performs the poorest (Cf. Figure 6). Nevertheless the gap is smaller than in (E3) and for high period times and $n \geq 20$, its latency is comparable to those of the other heuristics. For $n \geq 20$, **P4 3-Split bi** achieves for the first time the best results and the latency of **P2 2-Split bi** is only one time lower. When $n = 5$, **L2 2-Split bi** achieves the best latency, but the period values are not competitive with **P1 2-Split mono** and **L1 2-Split mono**, which obtain the smallest periods (for slightly higher latency times).

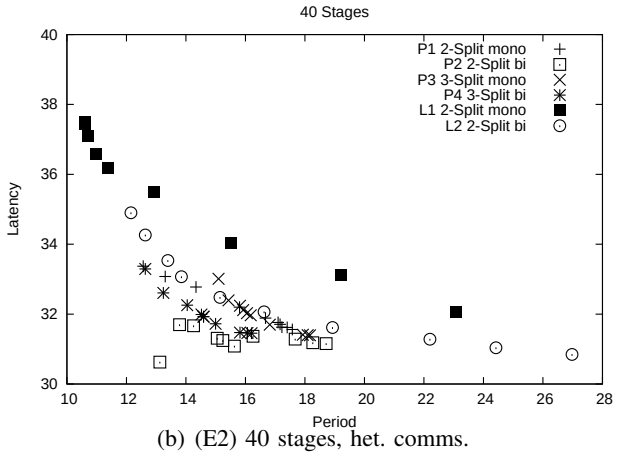
In Table I the **failure thresholds** of the different heuristics are shown. We denote by failure threshold the largest value of the fixed period or latency for which the heuristic was not able to find a solution. We state that **P1 2-Split mono** has the smallest failure thresholds whereas **P3 3-Split mono** has the highest values. Surprisingly the failure thresholds (for fixed latencies) of the heuristics **L1 2-Split mono** and **L2 2-Split bi** are the same, but their performance differs enormously as stated in the different experiments.

b) With $p = 100$ processors: Many results are similar with $p = 10$ and $p = 100$ processors, thus we only report the main differences. In particular, the failure thresholds table is not included since it displays similar results.

First we observe that both periods and latencies are lower with the increasing number of processors. This is easy to explain, as all heuristics always choose fastest processors first, and there is much more choice with $p = 100$. All heuristics keep their general behavior, i.e., their curve characteristics. But the relative performance of some heuristics changes dramatically. The results of **P3 3-Split mono** are much better, and we do get adequate

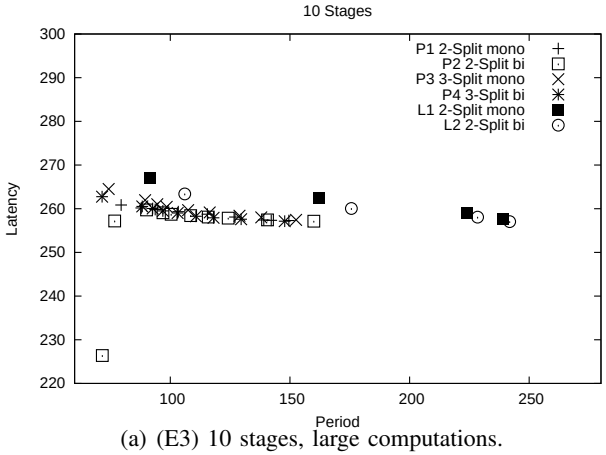


(a) (E1) 40 stages, hom. comms.

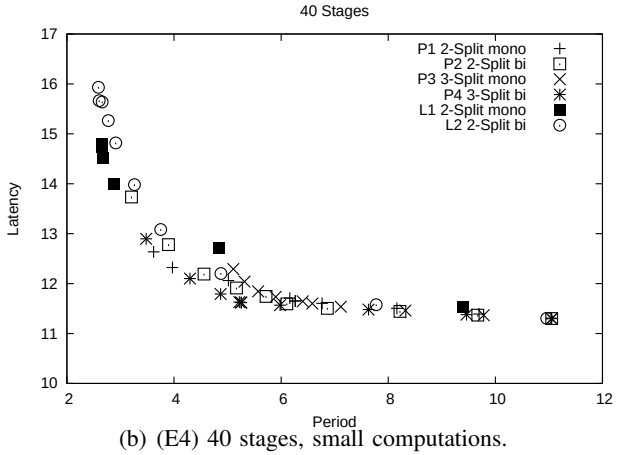


(b) (E2) 40 stages, het. comms.

Fig. 7. Extension to 100 processors, balanced communications/computations.



(a) (E3) 10 stages, large computations.



(b) (E4) 40 stages, small computations.

Fig. 8. Extension to 100 processors, imbalanced communications/computations.

latency times (compare Fig. 3(b) and 7(a)). Furthermore the multi-criteria heuristics turn out to be much more performing. An interesting example can be seen in Fig. 7(b): all multi-criteria heuristics outperform their mono-criterion counterparts, even **L2 2-Split bi**, which never had a better performance than **L1 2-Split mono** when $p = 10$.

In the case of imbalanced communications/computations, we observe that all heuristics achieve almost the same results. The only exception is the binary-search heuristic **P2 2-Split bi**, which shows a slightly superior performance as can be seen in Fig. 8(a). The performance of **P4 3-Split bi** depends on the number of stages. In general, **P1 2-Split mono** is better than **P4** when $n \leq 10$, but for $n \geq 20$, **P4** owns the second position after **L2 2-Split bi** and even performs best in the configuration small computations, $n = 40$ (see Fig. 8(b)).

c) Comparison to the LP solution: In order to assess the absolute performance of our heuristics, we compared their solution to the optimal solution returned by the linear program of Section V. We took the 500 set of parameters considered in (E2) with 40 stages, and set up a time limit to one hour to solve each problem instance with the LP solver. Within this time limit, 94 instances (out of 500) were solved for a fixed period, and only 8 for a fixed latency.

Comparing the solution returned by the heuristics to the optimal solution for those cases for which we obtained results, the conclusion is that the **2-Split** heuristics with fixed period (**P1** and **P2**) behave best, being on average at less than 5% of the optimal. In many cases, these heuristics return the optimal solution, and otherwise their solution is close to the optimal. The **3-Split** heuristics (**P3** and **P4**) are slightly behind, with 6%, thus still very competitive.

For **L1 2-Split mono** and **L2 2-Split bi**, we could not produce statistics because of the low number of solved instances, but we found the optimal solution in 5 of the 8 cases. In the three remaining cases, however, heuristic

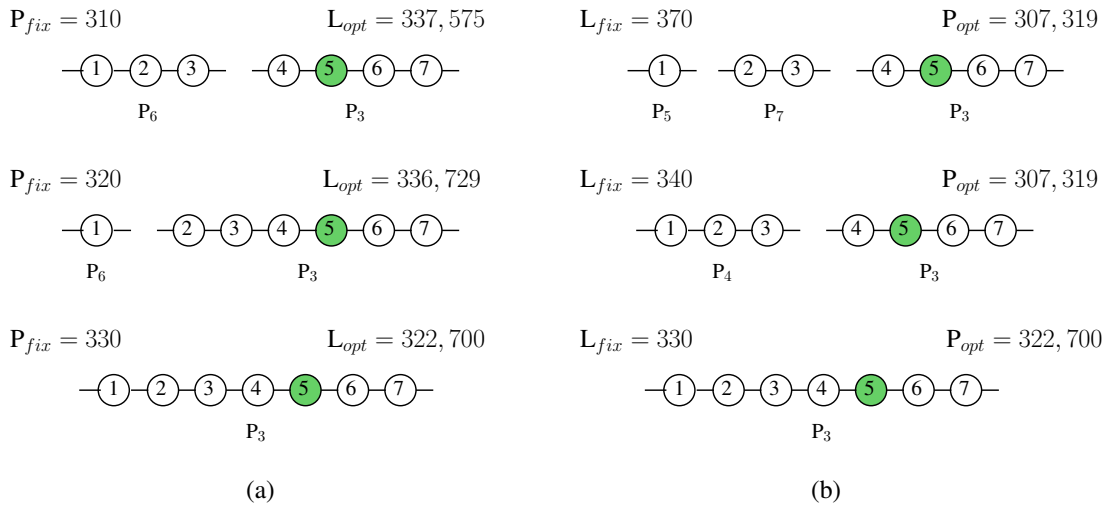


Fig. 9. LP solutions strongly depend on fixed initial parameters.

results are further from the optimal, leading to periods up to 60% away from the optimal solution. The detailed values of the LP and the heuristics can be found on the Web (see [17]).

3) Summary: Overall we conclude that the performance of bi-criteria heuristics versus mono-criterion heuristics highly depends on the number of available processors. For a small number of processors, the simple splitting technique which is used in **P1 2-Split mono** and **L1 2-Split mono** is very competitive as it almost always minimizes the period with acceptable latency values. The bi-criteria splitting **P2 2-Split bi** mainly minimizes latency values at the price of longer periods. Nevertheless depending upon the application, this heuristic seems to be very interesting, whenever small latencies are demanded. On the contrary, its counterpart **L2 2-Split bi** does not provide convincing results, and both **3-Split** heuristics do not achieve the expected performance. However, when increasing the number of available processors, we observe a significant improvement of the behavior of bi-criteria heuristics. **L2 2-Split bi** turns out to outperform the mono-criterion version and **P2 2-Split bi** upgrades its period times such that it outplays its competitors. Finally, both 3-Splitting heuristics perform much better and **P4 3-Split bi** finds its slot.

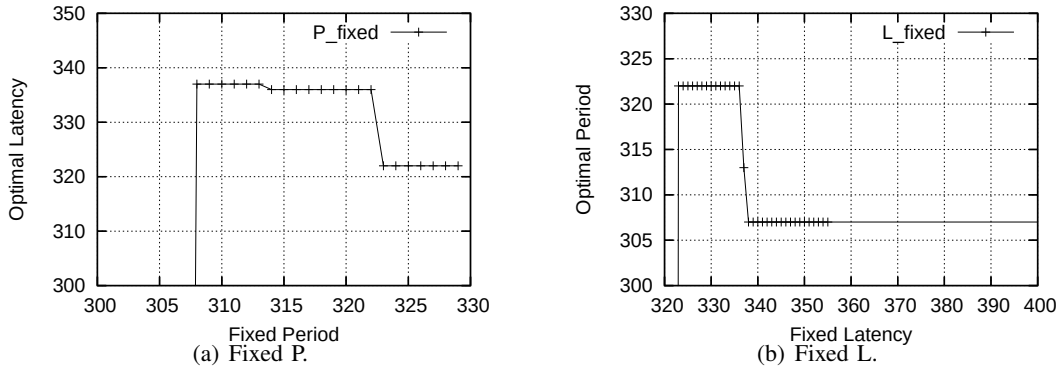
To conclude, we point out that in most of the cases for which we are able to compute the optimal solution thanks to the linear program, our heuristics are very close to this optimal solution (or even returning the optimal), in particular for the heuristics with a fixed period. Heuristics with a fixed latency are slightly further from the optimal. Note that this is a very promising result, given that running all heuristics for all parameters takes less than one minute, while the LP solver does not succeed to find the solution within one hour for most problem instances.

B. Experiments and Simulations for the JPEG encoder

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

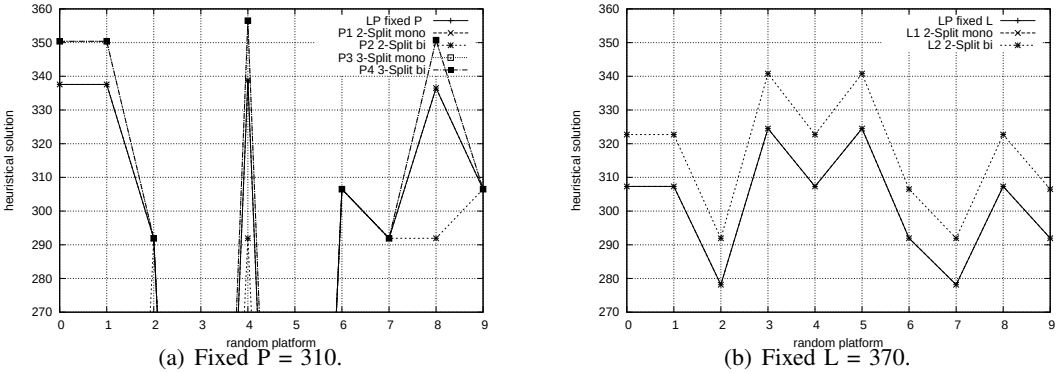
1) Influence of fixed parameters: In the first test series, we examine the influence of fixed parameters on the solution of the linear program. As shown in Fig. 9, the division into intervals is highly dependant of the chosen fixed value. The optimal solution to minimize the latency (without any additional 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. 9(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 (with fixed latency) can be found in Fig. 9(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. 9(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



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

Fig. 10. Bucket behavior of LP solutions.



(a) Fixed P = 310. (b) Fixed L = 370.

Fig. 11. Behavior of the heuristics (comparing to LP solution).

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. 10.

2) 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. 11. 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.

We first point out that **P2 2-Split bi** achieves good latency results, but the fixed period 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 solutions with a low latency, thus reaching the best latency/period ratios. Second, all solutions, LP and heuristics, always keep the stages 4 to 7 together (see Fig. 9 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 **P1 2-Split mono** always finds the optimal period for a fixed latency and we therefore recommend this heuristic for period optimization. In the case of period minimization for a fixed latency, then **L1 2-Split mono** 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.

3) 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. 12 compares the average of the theoretical results of the heuristics to the average simulative performance, and the simulative behavior nicely mirrors the theoretical behavior.

4) Summary: We conclude that the general behavior of the heuristics in the special case of the JPEG encoder pipeline mainly reflects the observations made in the general case. The 2-Split technique seems to be a very powerful mechanism for bi-criteria optimization with a small number of processors, and always returns the optimal solution for the JPEG encoder.

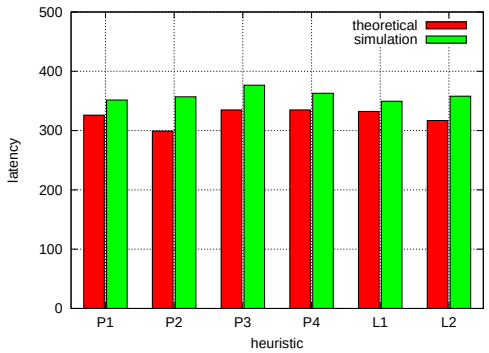


Fig. 12. MPIsimulation results - Simulative latency.

VIII. Related Work

As already mentioned, this work is an extension of the work of Subhlok and Vondran [12], [13] for pipeline applications on homogeneous platforms. We have extended their complexity results to heterogeneous platforms.

Several papers consider the problem of mapping communicating tasks onto heterogeneous platforms, but for a different applicative framework. In [18], Taura and Chien consider applications composed of several copies of the same task graph, expressed as a DAG (directed acyclic graph). These copies are to be executed in pipeline fashion. Taura and Chien also restrict to mapping all instances of a given task type (which corresponds to a stage in our framework) onto the same processor. Their problem is shown NP-complete, and they provide an iterative heuristic to determine a good mapping. At each step, the heuristic refines the current clustering of the DAG. Beaumont et al [19] consider the same problem as Taura and Chien, i.e., with a general DAG, but they allow a given task type to be mapped onto several processors, each executing a fraction of the total number of tasks. The problem remains NP-complete, but becomes polynomial for special classes of DAGs, such as series-parallel graphs. For such graphs, it is possible to determine the optimal mapping owing to an approach based upon a linear programming formulation. Due to complex mapping rules, the approach of [19] can only achieve optimal throughput by using very long periods: hence the simplicity and regularity of the schedule are lost, while the latency is severely increased. On the contrary, the simpler mapping rules used in this paper allow for better period/latency trade-offs.

Another important series of papers comes from the DataCutter project [20]. One goal of this project is to schedule multiple data analysis operations onto clusters and grids, decide where to place and/or replicate various components [21], [22], [23]. A typical application is a chain of consecutive filtering operations, to be executed on a very large data set. The task graphs targeted by DataCutter are more general than linear pipelines or forks, but still more regular than arbitrary DAGs, which makes it possible to design efficient heuristics to solve the previous placement and replication optimization problems.

A recent paper [24] targets generalized workflows structured as arbitrary DAGs, and considers an instance of the bi-criteria optimization problem where the latency is optimized under a fixed throughput constraint. Only completely homogeneous platforms are considered in [24]. It would be very interesting (but also very challenging) to extend the heuristics of [24] to heterogeneous frameworks, and to compare them with the heuristics that we specifically designed for pipeline workflows.

The former related works dealt mostly with generalized applications which assume synthetic workload and application pipelines. Our paper applies the bi-criteria mapping problem to a concrete application pipeline in digital image encoding. In this domain, parallelization strategies have been considered beforehand, but the pipelined nature of the applications has not been fully exploited yet.

Many authors, started from the blockwise independent processing of the JPEG encoder in order to apply simple data parallelism for efficient parallelization. This fine-grain parallelization opportunity is for instance exploited in [25], [26]. In addition, parallelization of almost all stages, from color space conversion, over DCT to the Huffman encoding has been addressed [27], [28]. Recently, with respect to the JPEG2000 codec, efficient parallelization of wavelet coding has been introduced [29]. 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 block-wise parallelization [30], [31]. 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. [32] 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.

IX. Conclusion

In this paper, we have studied a difficult bi-criteria mapping problem onto *Communication Homogeneous* platforms. We restricted ourselves to the class of applications which have a pipeline structure, and we have studied the complexity of the problem. While minimizing the latency is simple, the problem of minimizing the pipeline period is NP-hard, and thus the bi-criteria problem is NP-hard. We provided several efficient polynomial heuristics, either to minimize the period for a fixed latency, or to minimize the latency for a fixed period.

A typical application class, the digital image coding where images are processed in steady-state mode, has been considered as proof-of-concept. We have provided a detailed study of the bi-criteria mapping (minimizing period and latency) of the JPEG encoder application pipeline on a cluster of workstations. Experimental evaluations have then been carried out on generalized data sets and machine configurations, as well as for the JPEG encoder application.

The results of the general experiments show that the efficiency highly depends on platform parameters such as number of stages and number of available processors. Simple mono-criterion splitting heuristics perform very well when there is a limited number of processors, whereas bi-criteria heuristics perform much better when increasing the number of processors. Overall, the introduction of bi-criteria heuristics was not fully successful for small clusters but turned out to be mandatory to achieve good performance on larger platforms. Also, we have derived an integer linear program formulation of the bi-criteria optimization problem, which enabled us to assess the absolute performance of the heuristics. The results were quite satisfactory on those (not so many) instances for which we could compute the optimal solution in reasonable time. Finally, we observed that the 2-Split technique always returned the optimal solution for the JPEG encoder.

In future work, we plan to extend our heuristics to fully heterogeneous platforms, which appears to be challenging, even for a mono-criterion optimization problem. Indeed, because all link bandwidths are different, it seems hard to predict communication times as long as the mapping is not fully constructed. Thus, it is not easy to determine a strategy capable of simultaneously load balance computations while keeping communications under a prescribed threshold. In the longer term, we plan to perform real-world experiments on heterogeneous platforms, using an already-implemented skeleton library, in order to compare the effective performance of the application for a given mapping (obtained with our heuristics) against the theoretical performance of this mapping.

A natural extension of this work would be to consider other widely used skeletons. For example, when there is a bottleneck in the pipeline operation due to a stage which is both computationally-demanding and not constrained by internal dependencies (such as the FDCT stage of the JPEG encoder), we can nest another skeleton in place of this stage. For instance a farm or deal skeleton would allow to split the workload of the initial stage among several processors. Using such deal skeletons may be either the programmer's decision (explicit nesting in the application code) or the result of the mapping procedure. Extending our mapping strategies to automatically identify opportunities for deal skeletons, and implement these, is a difficult but very interesting perspective.

Acknowledgments. We thank the reviewers for their numerous comments and suggestions, which greatly improved the final version of the paper.

REFERENCES

[1] A. Benoit, V. Rehn-Sonigo, and Y. Robert, "Multi-criteria scheduling of pipeline workflows," in *HeteroPar'2007: International Conference on Heterogeneous Computing, jointly published with Cluster'2007*. IEEE Computer Society Press, 2007, pp. 515–524.

[2] ——, "Bi-criteria pipeline mappings for parallel image processing," in *ICCS'2008, the 8th International Conference on Computational Science*, ser. LNCS, vol. 5101. Springer Verlag, 2008, pp. 215–225.

[3] B. A. Shirazi, A. R. Hurson, and K. M. Kavi, *Scheduling and load balancing in parallel and distributed systems*. IEEE Computer Science Press, 1995.

[4] M. Cole, "Bringing Skeletons out of the Closet: A Pragmatic Manifesto for Skeletal Parallel Programming," *Parallel Computing*, vol. 30, no. 3, pp. 389–406, 2004.

[5] F. Rabhi and S. Gorlatch, *Patterns and Skeletons for Parallel and Distributed Computing*. Springer Verlag, 2002.

[6] P. Monnes and B. Furht, "Parallel JPEG Algorithms for Still Image Processing," in *Southeastcon'94. Creative Technology Transfer - A Global Affair. Proceedings of the 1994 IEEE*. IEEE Conference Proceeding, apr 1994, pp. 375 – 379.

[7] G. K. Wallace, "The jpeg still picture compression standard," *Commun. ACM*, vol. 34, no. 4, pp. 30–44, 1991.

[8] C. Wen-Hsiung, C. Smith, and S. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Tranfsorm," *IEEE Transactions on Communications*, vol. 25, no. 9, pp. 1004–1009, 1977.

[9] P. Bhat, C. Raghavendra, and V. Prasanna, "Efficient collective communication in distributed heterogeneous systems," in *ICDCS'99 19th International Conference on Distributed Computing Systems*. IEEE Computer Society Press, 1999, pp. 15–24.

[10] ——, "Efficient collective communication in distributed heterogeneous systems," *Journal of Parallel and Distributed Computing*, vol. 63, pp. 251–263, 2003.

[11] T. Saif and M. Parashar, "Understanding the behavior and performance of non-blocking communications in MPI," in *Proceedings of Euro-Par 2004: Parallel Processing*, ser. LNCS 3149. Springer, 2004, pp. 173–182.

[12] J. Subhlok and G. Vondran, "Optimal mapping of sequences of data parallel tasks," in *Proc. 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP'95*. ACM Press, 1995, pp. 134–143.

[13] ——, "Optimal latency-throughput tradeoffs for data parallel pipelines," in *ACM Symposium on Parallel Algorithms and Architectures SPAA'96*. ACM Press, 1996, pp. 62–71.

[14] A. Benoit, Y. Robert, and E. Thierry, "On the Complexity of Mapping Linear Chain Applications onto Heterogeneous Platforms," LIP laboratory, ENS Lyon, Research Report 2008-32, October 2008, available at http://graal.ens-lyon.fr/∼abenoit.

[15] A. Benoit and Y. Robert, "Mapping pipeline skeletons onto heterogeneous platforms," *J. Parallel Distributed Computing*, vol. 68, no. 6, pp. 790–808, 2008.

[16] A. Pinar and C. Aykanat, "Fast optimal load balancing algorithms for 1D partitioning," *J. Parallel Distributed Computing*, vol. 64, no. 8, pp. 974–996, 2004.

[17] Code for the heurisics and full set of results, available at http://graal.ens-lyon.fr/∼vsonigo/code/multicriteria.

[18] K. Taura and A. A. Chien, "A heuristic algorithm for mapping communicating tasks on heterogeneous resources," in *Heterogeneous Computing Workshop*. IEEE Computer Society Press, 2000, pp. 102–115.

[19] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert, "Assessing the impact and limits of steady-state scheduling for mixed task and data parallelism on heterogeneous platforms," in *HeteroPar'2004: International Conference on Heterogeneous Computing, jointly published with ISPDC'2004: Int. Symp. on Parallel and Distributed Computing*. IEEE Computer Society Press, 2004, pp. 296–302.

[20] "DataCutter Project: Middleware for Filtering Large Archival Scientific Datasets in a Grid Environment," http://www.cs.umd.edu/projects/hpsl/ResearchAreas/DataCutter.htm.

[21] M. Beynon, A. Sussman, U. Catalyurek, T. Kurc, and J. Saltz, "Performance optimization for data intensive grid applications," in *Proceedings of the Third Annual International Workshop on Active Middleware Services (AMS'01)*. IEEE Comp. Soc. Press, 2001.

[22] M. D. Beynon, T. Kurc, A. Sussman, and J. Saltz, "Optimizing execution of component-based applications using group instances," *Future Generation Computer Systems*, vol. 18, no. 4, pp. 435–448, 2002.

[23] M. Spencer, R. Ferreira, M. Beynon, T. Kurc, U. Catalyurek, A. Sussman, and J. Saltz, "Executing multiple pipelined data analysis operations in the grid," in *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002, pp. 1–18.

[24] N. Vydyanathan, U. Catalyurek, T. Kurc, P. Saddayappan, and J. Saltz, "An approach for optimizing latency under throughput constraints for application workflows on clusters," Ohio State University, Columbus, OH, Research Report OSU-CISRC-1/07-TR03, Jan. 2007, available at ftp://ftp.cse.ohio-state.edu/pub/tech-report/2007.ShortversionappearsinEuroPar'2008.

[25] J. Falkemeier and G. Joubert, *High Performance Computing: Technologies, Methods and Applications*, ser. Advances in Parallel Computing. North Holland, 1995, no. 10, ch. Parallel image compression with JPEG for multimedia applications, pp. 379–394.

[26] K. Shen, G. Cook, L. Jamieson, and E. Delp, "An overview of parallel processing approaches to image and video compression," in *Image and Video Compression*, M. Rabbani, Ed., vol. Proc. SPIE 2186. SPIE, 1994, pp. 197–208.

[27] L. V. Agostini, I. S. Silva, and S. Bampi, "Parallel color space converters for JPEG image compression," *Microelectronics Reliability*, vol. 44, no. 4, pp. 697–703, 2004.

[28] T. Kumaki, M. Ishizaki, T. Koide, H. J. Mattausch, Y. Kuroda, H. Noda, K. Dosaka, K. Arimoto, and K. Saito, "Acceleration of DCT Processing with Massive-Parallel Memory-Embedded SIMD Matrix Processor," *IEICE Transactions on Information and Systems - LETTER- Image Processing and Video Processing*, vol. E90-D, no. 8, pp. 1312–1315, 2007.

[29] P. Meerwald, R. Norcen, and A. Uhl, "Parallel JPEG2000 Image Coding on Multiprocessors," in *IPDPS'02, Proceedings of the 16th International Parallel and Distributed Processing Symposium*. IEEE Computer Society Press, 2002, p. 248.

[30] M. Ferretti and M. Boffadossi, "A Parallel Pipelined Implementation of LOCO-I for JPEG-LS," in *17th International Conference on Pattern Recognition (ICPR'04)*, vol. 1. IEEE Computer Society Press, 2004, pp. 769–772.

[31] M. Papadonikolakis, V. Pantazis, and A. P. Kakarountas, "Efficient high-performance ASIC implementation of JPEG-LS encoder," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE2007)*, vol. IEEE Comm. Soc. Press, 2007, pp. 159–164.

[32] S. L. Shee, A. Erdos, and S. Parameswaran, “Architectural Exploration of Heterogeneous Multiprocessor Systems for JPEG,” *International Journal of Parallel Programming*, vol. 35, pp. 140–162, 2007.