Scheduling and Energy-Distortion Tradeoffs with Operational Refinement of Image Processing

Davide Anastasia and Yiannis Andreopoulos
Dept. of Electronic and Electrical Engineering
University College London

Abstract— Ubiquitous image processing tasks (such as transform decompositions, filtering and motion estimation) do not currently provide graceful degradation when their clock-cycles budgets are reduced, e.g. when delay deadlines are imposed in a multi-tasking environment to meet throughput requirements. This is an important obstacle in the quest for full utilization of modern programmable platforms’ capabilities, since: (i) worst-case considerations must be in place for reasonable quality of results; (ii) throughput-distortion tradeoffs are not possible for distortion-tolerant image processing applications without cumbersome (and potentially costly) system customization. In this paper, we extend the functionality of the recently-proposed software framework for operational refinement of image processing (ORIP) and demonstrate its inherent throughput-distortion and energy-distortion scalability. Importantly, our extensions allow for such scalabilities at the software level, without needing hardware-specific customization. Extensive tests on a mainstream notebook computer and on OLPC’s subnotebook ("xo-laptop") verify that the proposed designs provide for: (i) seamless quality-complexity scalability per video frame; (ii) up to 60% increase in processing throughput with graceful degradation in output quality; (iii) up to 20% more images captured and filtered for the same power-level reduction on the xo-laptop.

Keywords: software realizations of image processing; programmable platforms; incremental refinement of computation; energy-distortion scalability; throughput-distortion scalability

I. INTRODUCTION

Current state-of-the-art multimedia-enabled portable systems (laptops, subnotebooks, mobile phones, etc.) offer very little synergy between the system layer (software design, processor, task manager) and the multimedia application layer (e.g. image processing task, such as filtering). For example, if one is watching a movie on a portable video player and this is draining the system resources (battery), current systems do not allow for seamless tradeoffs in visual quality vs. battery life (execution time per task). Similarly, if the processing throughput (in frames-per-second) needs to be increased in order to achieve real-time performance in a multitasking environment, current multimedia systems do not allow for throughput-distortion tradeoffs. In such cases, the user is practically facing the all-or-nothing approach of digital systems, while one would strongly opt for a best-effort approach, often found in analog systems, where energy autonomy or processing throughput would be increased with graceful degradation in the processed video quality.

Several works have tried to address this problem; they can be broadly separated in two categories: system-specific designs, where complexity-distortion frameworks have been proposed based on platform-specific customization [1][2], and algorithm-specific designs, where customizations of particular algorithms are proposed to reduce their requirements and provide a “reduced-complexity” profile. Examples of this category are reduced-complexity video coding [3] or reduced-complexity approximate transform computations [4][5].

In the first category, significant effort is required to create customized realizations for each system of interest, which may not come at a justifiable cost. Furthermore, the derived implementations are usually difficult to adjust to the dynamic resource profile required by the user or the system task scheduler (e.g. on-the-fly scaling of the processing throughput in frames-per-second, or switching to low-power modes to prolong battery life). In the second category, numerous algorithmic profiles (e.g. skipping transform coefficients, reduced-precision filtering, or reduced-precision motion estimation) are required to provide fine-grain quality-complexity scalability. However, once the desired operational profile is chosen for the algorithm, a fixed realization is produced that does not allow for arbitrary termination of each video frame processing when a scheduler reduces the clock cycle budget.

Recently, a software framework for incremental computation of image processing operations has been proposed [6]. The derived framework, termed operational refinement of image processing (ORIP) [7], is performing progressive computation of image processing tasks by construction, i.e. without the need for algorithm-specific or hardware-specific modifications apart from standard compiler optimization. ORIP can perform any integer-to-integer linear image processing task incrementally and provide increased output quality for increased resources [6]. It can be straightforwardly ported onto any platform (e.g. laptop, subnotebook, mobile phone) with a C++ compiler and provides for quality-complexity scalability without requiring system-specific customization. Image filtering, cross-correlation and motion estimation have already been included in ORIP [6].

In this paper, we extend ORIP by including transform decompositions and propose and test an application framework demonstrating ORIP’s capabilities in real-world applications. For this purpose, we build a scheduling framework using the cross-platform OpenMP framework [8] and then test the capabilities of the final solution in two diverse real-world platforms: a mainstream notebook computer and the ultra low-power subnotebook of the OLPC foundation (xo-laptop) [9]. Section II outlines ORIP and presents the case of block transform decomposition and reconstruction not covered in previous work. Section III presents the proposed modifications.
as well as the implemented scheduling framework. Section IV presents the energy-distortion scalability obtained with the proposed framework within a capturing and filtering application on the xo-laptop, and Section V concludes the paper.

II. OPERATIONAL REFINEMENT OF IMAGE PROCESSING

Consider an input video frame $I$ consisting of 8-bit non-negative input pixels. Transform decompositions and filtering are based on block-by-block processing, hence the first stage of ORIP is to consider consecutive input blocks $B_0, \ldots, B_M$ with $M$ indicating the packing capability defined in [6]. For example, for filtering operations, these are $M$ partially-overlapping "stripes" of image pixels [6].

A. ORIP Framework Operation

The processing starts by splitting all $m = \{1, \ldots, M\}$ input blocks into biplanes, $B_{m,\text{bit}}$, from the $(N - 1)$th most significant biplane (MSB) to the 0th least significant biplane (LSB), with $N = 8$ for each input image\(^1\). The following steps occur for all increment layers $n$, from $n = N - 1$ (MSB) to $n = 0$ (LSB) [6]:

1) Packing: For each layer $n$, a single block is formed from all $M$ blocks by packing them together, via:
   \[
   D_{n,\text{bit}} = \sum_{m=1}^{M} B_{m,\text{bit}} e^{\varepsilon^{m-1}},
   \]
   with $\varepsilon$ an appropriate packing coefficient.

2) Processing: The image processing operation (transformation, convolution) uses $D_{n,\text{bit}}$ and produces a resulting block $R_{n,\text{bit}}$.

3) Unpacking: The results can be unpacked sequentially by the following process:
   \[
   m = 1 : \quad U_{1,\text{bit}} = \left[R_{n,\text{bit}} + 0.5\right]
   \]
   \[
   \forall m \in \{2,M\} : \quad \varepsilon^{m} R_{m,\text{bit}} \leftarrow \frac{1}{\varepsilon^{m}} (R_{m,\text{bit}} - U_{m-1,\text{bit}})
   \]
   \[
   U_{n,\text{bit}} = \left[R_{n,\text{bit}} + 0.5\right]
   \]

4) Incrementation: After unpacking, the final stage of the incremental computation augments the previously-computed results of increment layers $N - 1, \ldots, n + 1$ of all $M$ blocks by adding to them the results of the current layer, $U_{n,\text{bit}}$, and the results are added to the previous ones by (4).

Even though the description focuses on $N$ increment layers, in practice biplanes can be grouped together to form a single increment layer [6]. The process described above is guaranteed to produce the correct results if all blocks $B_{m,\text{bit}}$ are packed in a sufficiently sparse manner via multiplications with $\varepsilon^{m-1}$. As shown by the analysis of [6], the values of the operational parameters $M$ and $\varepsilon$ depend on the dynamic range of the computation and can be derived a priori if the algorithm’s parameters (filtering or transform kernel) are known. Initial experiments [6] demonstrate that 64-bit floating-point representation can offer higher speed in comparison to integer representation, as most processors have highly-optimized arithmetic units for floating-point operations. In addition, floating-point representations can preserve the sign of each packed operation.

B. Incremental Transform Decomposition and Reconstruction

The input of this case is $Q$ groups of $M$ blocks of $C \times C$ input pixels, $B_{q,n}^0$, $1 \leq m \leq M$, with $C = \{4,8,16\}$ for typical cases of block transforms found in the literature [10][11]. The transform matrix is given by a $C \times C$ integer kernel $T_{for}$, e.g. the H.264 $4 \times 4$ transform [10]. Transform kernels with non-integer coefficients can be approximated by a fixed-point (FXP) representation with the appropriate number of fractional bits. Hence, they can be computed with an integer kernel followed by inverse scaling after the termination of the calculation and can be accommodated by our framework. The following describe the proposed incremental computation for all $M$ blocks of each group of blocks $q$, $1 \leq q \leq Q$.

Under an integer transform kernel, the decomposition of the $m$th block is performed by:

\[
\forall m \in \{1,M\} : \quad U_{n,\text{full}} = T_{for}^n B_{m,\text{full}} T_{for}^T.
\]

When biplanes of the input are used for the transform decomposition, the process can be performed for each biplane $n$ (from $n = N - 1$ to $n = 0$) of the $m$th block by:

\[
\forall m \in \{1,M\} : \quad U_{m,\text{bit}} = T_{for} B_{n,\text{bit}} T_{for}^T.
\]

and the results are added to the previous ones by (4).

The above process was already proposed within transform-specific formulations for incremental computation of the discrete Fourier transform [12] and the lifting-based discrete wavelet transform [13]. Here, we consider packing the results in order to accelerate the incremental computation in software. We form $D_{n,\text{bit}}$ by (1) and it is used to compute the packed result of all $M$ blocks by:

\[
R_{n,\text{bit}} = T_{for} D_{n,\text{bit}} T_{for}^T.
\]

The results are unpacked from $R_{n,\text{bit}}$ using (2) and (3) and the final results per biplane $n$ are derived by (4). Notice that only a single block transform decomposition with block $D_{n,\text{bit}}$ is performed by (7) instead of the $M$ block decompositions performed by (6). Hence, we expect to save operations by combining blocks together via the incremental packing approach of ORIP. The total number of blocks combined (packing capability), $M$, depends on the worst-case dynamic range $A_{\text{max}}$ of (6), as shown in previous work [6]. This range can be found by assuming the worst-case input block:

\[
0 \leq i, j < C, \quad q = \{0,1\} : \quad S_q[i,j] = \begin{cases} 1, & \text{if } (-1)^q T_{for} [i,j] > 0 \\ 0, & \text{if } (-1)^q T_{for} [i,j] \leq 0 \end{cases}
\]

\[
A_{\text{max}} = \max_{q} \max_{i,j} \left\{ \sum_{i,j=0}^{C-1} T_{for} [i,j] \cdot S_q[i,j] \right\},
\]

Concerning the transform reconstruction (synthesis) process, the same approach can be followed, where the synthesis is given by

\[1\] We follow the notational conventions of [6]. Boldface capital letters indicate matrices; the corresponding italicized letters indicate matrix elements; all indexes are integers; superscripts in matrices or scalars indicate the biplane number. Notation $A_{\text{bit}}$ indicates $A$ computed using the $n$th biplane of the input image; while $A_{\text{full}}$ indicates $A$ computed using input biplanes $N, N - 1, \ldots, n$.\]
\[ \forall m \in \{1, M\} : B_{m, \text{full}}^0 = T_{\text{inv}}^0 R_{m, \text{full}}^0 T_{\text{inv}}^0, \]  
\[ (10) \]
with \( T_{\text{inv}}^0 = T_{\text{inv}}^1 \). In this case the maximum bitplane of the input is changed depending on the dynamic range expansion of the forward transform. When using packing with integer representation, the incremental approach as presented so far only covers the use of transform kernels with non-negative coefficients because the sign information is not preserved via integer packing. In order to cover the general case of arbitrary transform kernels, we need to convert all transform coefficients to non-negative numbers by:

\[ T_{\text{for}}^+ = T_{\text{for}} + P, \]  
\[ (11) \]
and \( I_{C \times C} \) a \( C \times C \) matrix of ones. After the incrementally-computed decomposition is performed for each input block \( D_{\text{bit}}^n \) using \( T_{\text{for}}^+ \), we need to compensate for the added component of the kernel of \( (11) \) during the derivation of the final results per bitplane. However, simple linear algebra shows that several multiplications and additions are needed in order to derive the correct result since the decomposition with the transform kernel of \( (11) \) derives:

\[ R_{\text{bit}}^{n,+} = T_{\text{for}} D_{\text{bit}}^n T_{\text{for}}^T + P D_{\text{bit}}^n T_{\text{for}}^T + T_{\text{for}} D_{\text{bit}}^n P^T + P D_{\text{bit}}^n P^T, \]  
\[ (12) \]
out of which only the term \( T_{\text{for}} D_{\text{bit}}^n T_{\text{for}}^T \) is the desired increment. Hence, we do not investigate this option in this paper and restrict our approach to floating-point.

### III. PROPOSED MODIFICATIONS – SCHEDULING EXPERIMENTS

Even though the ORIP framework has been tested for its execution vs. precision characteristics in previous work [6], its effects in practical scheduling and energy-distortion scalability are only speculated. In the next two subsections, we build upon the previously-proposed framework and derive some modifications that are tested via real application results.

#### A. General Modifications

The description of Section II.A indicates that all image blocks are packed together (i.e. the algorithm splits the image into \( M \) blocks). However, as shown by Subsection II.B, there are practical cases where the desired number of blocks is larger than the value of \( M \), e.g. in a 4x4 block transform decomposition of a 352x288-pixel image. For those cases, after packing \( Q \) groups of \( M \) blocks (where \( Q \times M \) gives the total number of image blocks), we interleave the processing, unpacking and incrementation processes. This means that, within each increment layer, steps 2–4 of Section II.A are applied in each group of blocks before moving to the next one. This interleaving allows for arbitrary termination of the algorithm even *in-between increment layers*. Hence, this feature provides for virtually seamless quality improvement with increased computation.

In the case of transform decompositions, once the processing is terminated using the packed block \( D_{\text{bit}}^n \), unpacking and incrementation of the results of all \( M \) blocks takes place. For filtering operations, unpacking and incrementation takes place after each individual application of the filtering kernel in block \( D_{\text{bit}}^n \). This allows for interrupting the process in case termination at a desired percentage of the output result of an increment layer is requested, or when an interrupt flag is raised by a task scheduler. The latter is discussed in detail in Subsection B. Notice that, once the first increment layer is fully completed, an initial result is available for the entire video frame. Hence, terminating at an arbitrary point after that will not leave any areas within the frame without any processing.

#### B. Scheduling

Conventional (non-incremental) real-time software for image processing tasks operates under worst-case assumptions, e.g. see [14]. Here, we want to investigate what happens when scheduling deadlines do not comply with the worst case. To this end, we consider the scenario where, for each video frame, the image processing task of interest is controlled by a scheduler (timer), which terminates the task after the scheduled time per video frame elapses. When the termination signal is received, the task *immediately* provides the already computed results for the input frame, before proceeding to the next video frame. We illustrate this approach in Figure 1.

In order to implement this design, we have used the cross-platform OpenMP framework [8] where two independent threads (timer and main thread) are concurrently executed. The two threads share the common memory element \( \text{flag\_int} \) to realize the signaling: when \( \text{flag\_int} \) is set to true by the timer thread, the application thread terminates the processing of the current frame and resets \( \text{flag\_int} \) to false.

In our first experiment, the termination signal is generated by the timer thread using an average value \( J \) with \( D \% \) of variability around the average value. Two cases are considered: (i) “regular-variability” scheduling, where \( J=100\% \) of the average frame completion time for each task and \( D=30\% \) of \( J \), and (ii) “aggressive-variability” scheduling, where \( J=80\% \) of the average frame completion time for each task and \( D=50\% \) of \( J \). In order to report results for both conventional and incremental versions of the algorithms, we measure two aspects: (i) the percentage of *uncovered frames*; these are frames with areas within them that have not been processed (covered) at all (i.e. areas with no decomposition or filtering, or no motion estimation for some blocks); (ii) the percentage of *fully-completed frames*; these are fully-covered frames *and* with the result computed at full precision. Naturally, for optimal performance, the first percentage should be as close to zero as possible, while the second should be as close to 100% as possible.

The results are given in Table 1. For all the experiments of this section, we used a Dell Latitude D630 notebook with an Intel Core 2 Duo processor (at 2.5GHz with 2Gb RAM). All programs were written in C++ and compiled with the Microsoft Visual Studio 2008 compiler in Windows XP, with all default optimizations of “-o2 (maximize speed)” mode. To achieve stable timing measurements with high precision, we used the Windows QueryPerformanceCounter() function and run all programs in highest priority. We chose random kernels for the cross-correlation and transform decomposition experiments (with sizes mentioned in the caption of Table 1), except where indicated. The 16x16 block matching used search range of ±16 pixels and realizes the algorithm described in [6]. For all experiments, the input was CIF-resolution video clips (Y channel only) and we split the 8 input bitplanes into \{3,3,2\} (3 increment layers) [6].
Figure 1. Time-driven computation of image processing tasks. The timer thread sends the stop signal to the application thread in order to terminate their execution for each frame. The application thread initiates the timer thread by the restart signal. The signaling is achieved via checking and setting/resetting flag_int.

It is important to remark that the proposed approach has an intermediate category of output results, which is covered frames but not fully-completed, i.e. not all increments have been computed. Representative visual examples of the observed artifacts are given in Figure 2. Post-processing with error concealment could potentially reduce the distortion caused by uncovered areas in both conventional and incremental processing at the cost of additional complexity. However, the results of Table 1 show that the proposed incremental approach rarely requires this, since, even under aggressive scheduling, the percentage of uncovered frames remains well below 1% in all but one experiment. This is an order of magnitude difference with the conventional approach that typically leaves more than 10% of the frames with uncovered areas when operating under scheduling. This demonstrates that, unlike the conventional implementations, the proposed approach obtains reasonable quality even when the scheduler does not provide for the worst-case.

It is interesting to observe that, apart from this advantage, the proposed method provides significantly-higher percentage of fully-completed frames under both scheduling provisions. We observed that the execution time of the proposed incremental approach fluctuates less across different frames in comparison to the conventional approach. This allows for successful completion of more frames when the scheduling time fluctuates around the mean execution time.

<table>
<thead>
<tr>
<th>Transform Decomposition</th>
<th>Regular-variability (J=100% of each method, D=30%)</th>
<th>Aggressive-variability (J=80% of each method, D=50%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scheduling Type</strong></td>
<td><strong>Uncovered</strong></td>
<td><strong>Fully-completed</strong></td>
</tr>
<tr>
<td>Conventional</td>
<td>33.90%</td>
<td>66.10%</td>
</tr>
<tr>
<td>Incremental</td>
<td>0.19%</td>
<td>91.06%</td>
</tr>
<tr>
<td><strong>2D Filtering</strong></td>
<td><strong>Uncovered</strong></td>
<td><strong>Fully-completed</strong></td>
</tr>
<tr>
<td>Conventional</td>
<td>4.41%</td>
<td>95.59%</td>
</tr>
<tr>
<td>Incremental</td>
<td>0.06%</td>
<td>99.87%</td>
</tr>
<tr>
<td><strong>Block Matching</strong></td>
<td><strong>Uncovered</strong></td>
<td><strong>Fully-completed</strong></td>
</tr>
<tr>
<td>Conventional</td>
<td>10.89%</td>
<td>89.11%</td>
</tr>
<tr>
<td>Incremental</td>
<td>0.28%</td>
<td>95.44%</td>
</tr>
</tbody>
</table>

Table 1. Percentage of uncovered and fully-completed frames for: 4x4 AVC transform [10] and 16x16 integer block transforms (top part); 8x8 cross-correlation and 12x12 Gaussian convolution (middle part); 8x8 and 16x16 block matching (bottom).
Figure 2. Visual example of video frame; from left to right: fully-completed frame, uncovered frame, covered frame but not fully completed (i.e. the result is not computed to full precision).

Figure 3: (a) Experimental results with reduced scheduling deadline; (b) SNR per frame for constant-time execution versus constant-quality execution (100 frames).

In a second scheduling experiment, we want to explore the throughput/quality tradeoffs enabled by the proposed approach via execution with fixed deadline per frame. Figure 3(a) shows average signal-to-noise (SNR) vs. throughput results (in terms of frames-per-second – fps) obtained with the incremental 2D convolution with the 12x12 Gaussian mask. SNR was measured per frame against the result of the conventional implementation operating at full precision. We gradually decreased the scheduling deadline (without variability) from $J = 31\text{ms}$ to $J = 19\text{ms}$ per frame, which leads to a 62.8% increase throughput, from 32.3fps to 52.6fps respectively, with a corresponding drop in SNR from infinity (identical results to full precision) to 19.36dB. It is important to remark that, for all results reported in Figure 3, no uncovered frames were produced. This straightforward quality-complexity scalability provides a very efficient tool when the processing requirements need to be scaled on-the-fly to match throughput demands.

Notice that constant-time execution is complementary to bitplane-based execution (shown in the results of [6]) where constant-quality processing is achieved but the execution time per frame can vary. The SNR results per frame shown in Figure 3(b) demonstrate this difference; there, the constant-quality execution was terminated at bitplane $n = 2$ per frame, while the constant-time execution imposed $J = 24\text{ms}$ per frame (41.7fps); both methods required virtually the same average time per frame (the difference was within a 5% margin). However, constant-time execution produces occasional drops in SNR in certain frames, while constant-quality execution provides near-constant SNR with occasional bursts of execution time.

IV. ENERGY-DISTORTION EFFICIENCY OF ORIP FOR REAL-TIME IMAGE CAPTURING AND PROCESSING ON THE XO-LAPTOP

In this experiment, we use the on-board camera of the xo-laptop to capture still images in raw YCbCr format (640x480 pixels) and apply the 8x8 cross-correlation algorithm with floating-point packing using a high-pass filter kernel. This corresponds to an image capturing and filtering scheme for edge detection in a live monitoring application. We used the Linux Hardware Abstraction Layer\textsuperscript{2} to periodically read the xo-laptop’s battery status during the algorithm execution. Our goal is to measure the power-level reduction when computing the high-pass filtering with different accuracies (in terms of terminating bitplane $n$). To this end, we switched the xo-laptop to terminal mode and converted the monitor to low-power (reflective) mode [9]. Image capturing was realized with the gstreamer tool [15] and ORIP was compiled with gcc4.0.0 (-o2) under Fedora Linux. We run the image capturing and filtering algorithm continuously from battery power level 97\% down to power level 17\%. Three increment layers were used for this experiment: the first processes bitplanes $\{7, 6, 5\}$ (terminating bitplane $n = 5$); the second processes bitplanes $\{4, 3, 2\}$ (terminating bitplane $n = 2$); and the third processes bitplanes $\{1, 0\}$ (terminating bitplane $n = 0$). Typical output results are shown in Figure 4 in terms of battery power level and number of images captured and processed for conventional full-precision processing and incremental processing with different terminating bitplanes.

\textsuperscript{2} http://www.freedesktop.org/wiki/Software/hal
The results demonstrate that the proposed software achieves up to 20% more images captured and processed for the same reduction in battery power level when reducing precision from \( n = 0 \) to \( n = 5 \). Conversely, once the experiment passes the 700th image, significant difference in power level can be observed for the same number of images in all approaches. For example, for 900 images, the power level goes from 28% for the conventional and the incremental approach with \( n = 0 \), to 35% for the incremental approach with \( n = 2 \), and to 41% for \( n = 5 \). For a low-power surveillance and monitoring scenario where for most of the time no activity is observed and hence the system can predominantly operate in lowest-precision mode, this indicates that there is potential of offering increased energy autonomy without requiring any specific customizations. We are confident that additional techniques, such as platform-specific optimization to reduce the power consumed by the image capturing process, could provide further improvements, since we observed that the capturing alone required most of the processing time per image (16% more than the full-precision conventional computation and 280% more than the lowest-precision incremental approach).

![Figure 4. Measured battery power-level reduction versus images captured and processed with different precision (different terminating bitplanes \( n \)) in the xo-laptop. The dotted line indicates the power-level reduction when operating the frame capturing and battery monitoring only (without any processing).](image)

V. CONCLUSION

The scheduling results of this paper demonstrate that, by exploiting the incremental nature of the operation refinement of image processing (ORIP) framework, the worst-case (“digital world”) approach of: ‘Can this image processing task be performed in \( X \) frames-per-second?’ changes to the best-effort (“analog world”) approach of: ‘What is the achieved quality when this task is performed in \( X \) frames-per-second?’ Similarly, the power-distortion results demonstrate that ORIP’s incremental computation allows for seamless prolongation of the battery life of a low-power device with a simple change of output quality level. Future research on better exploiting the packing capabilities of certain integer or floating-point representations, as well as system-specific customizations of the available software [7], such as computing using the graphics processing units, may permit further improvements in throughput and energy scalability with reduced output precision.

REFERENCES