Operational Refinement of Image Processing: ORIP$_{\mathrm{v1.0}}$

Davide Anastasia and Yiannis Andreopoulos[*]

Dept. of Electronic and Electrical Engineering, University College London, UK

Abstract: ORIP is a software framework that, in its current version, realizes incremental computation of transform decompositions, 2D convolution and frame-by-frame block matching. The uniqueness of ORIP is that it processes each input frame progressively and the results are produced with increased precision for increased processing time. We outline our motivation for creating ORIP, discuss its structure and main parameters, and present some indicative results.

1 Introduction

ORIP is a software-based approach to calculate a predefined image processing algorithm progressively: additional processing leads to better quality of the result per input image in a near seamless manner.

1.1. Motivation for Creating ORIP

We have witnessed a significant volume of research on multimedia streaming to bandwidth-constrained wireless (portable) devices, e.g. see [1] for a summary. However, currently there is very little work addressing the synergy between the *system layer* (software design, processor, task manager) and the *multimedia application layer* (e.g. image processing task, such as filtering). As a result, even though existing wireless infrastructure technology may provide for sufficient bandwidth for high framerate/high resolution video transmission, the device capabilities (e.g. battery life) remain a significant bottleneck. For example, if one is watching a streamed movie on a multimedia-enabled mobile phone 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). In such cases, the user is practically facing the on/off approach of digital systems, while one would strongly opt for a best-effort approach, often found in analogue systems, where energy autonomy would be increased with graceful degradation in the decoded video quality. We created ORIP as an early prototype that systematically addresses this challenge.

1.2. Overview

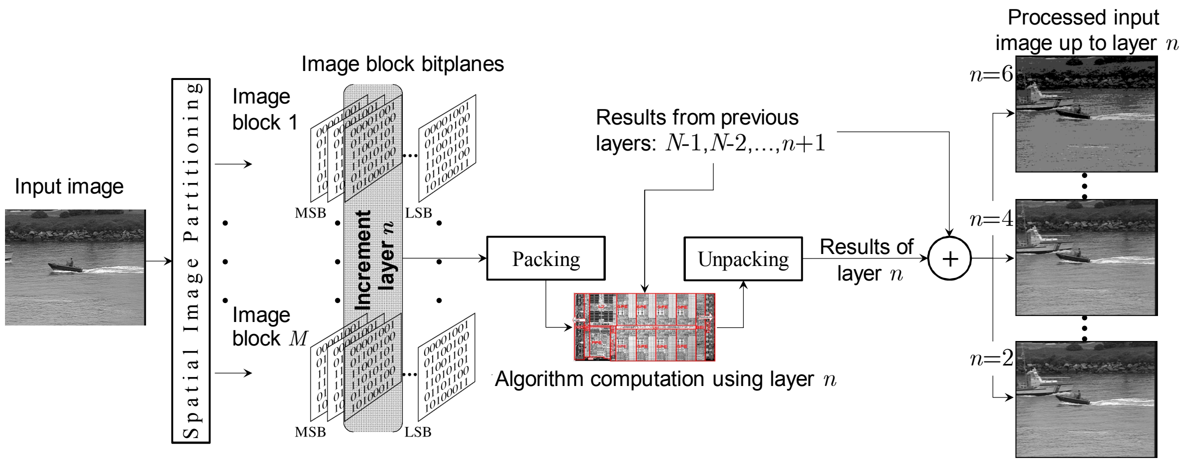
ORIP achieves progressive computation of image processing algorithms commonly used in coding systems based on the combination of bitplane-based computation [2][3] with a recently-proposed packing approach [4] that enables the calculation of multiple limited dynamic-range integer operations via one 32-bit or 64-bit arithmetic operation. Based on the derived design, ORIP is capable of stopping *virtually any time* during the frame processing and providing the result up to the computed precision. This provides for truly complexity-distortion scalable computation with a linear relationship between the required processing (execution cycles, drop in battery power level, or memory accesses) and achieved fidelity (distortion) measured via signal-to-noise ratio (SNR) or peak-signal-to-noise ratio (PSNR). A detailed description of ORIP is given in [5]. The current version (v1.0) is also available online for non-profit purposes [6].

2. Framework Structure

A general depiction of the proposed framework for incremental computation based on source refinements is presented in Figure 1. An input image is initially partitioned into M non-overlapping areas (stripes or blocks), whose binary (bitplane-by-bitplane) representation is shown in the middle of the figure, from most-significant bitplane (MSB) to the least-significant bitplane (LSB). A total of N *increment layers* are formed by grouping together the bits of all blocks belonging to the same bitplane. An example of an increment layer is shown pictorially in the shaded area. Each increment layer n

[*] This work was supported by EPSRC, grant no. EP/F020015/1.

($0 \leq n < N$) is also a layer of computation and we calculate the results of M blocks of each layer together using an incremental packing approach. For 8-bit images currently handled by ORIP, $N = 8$.



Input image
Spatial Image Partitioning
Image block 1
Image block M
Image block bitplanes
MSB
LSB
Increment layer n
Packing
Unpacking
Results from previous layers: N-1,N-2,...,n+1
Algorithm computation using layer n
Results of layer n
+
Processed input image up to layer n
n=6
n=4
n=2

Figure 1. Incremental refinement of computation using packing and unpacking of increment layers extracted progressively from the input image data. The output result is progressively refined via the computation of more increment layers.

After the packing of each increment layer, the desired image processing task is applied to the packed representation. Depending on the algorithm of interest, one can localize the calculation of the processing task around areas of interest based on the previously-computed increment layers. The produced result for each packed input is unpacked and the outputs are recovered for each layer. Then, the previously-computed results of previous layers are incremented by adding to them the results of the current layer. Processing and unpacking for each set of input blocks is interleaved such that, if at any point a termination flag is raised (e.g. via a scheduler), the already computed result can be provided immediately. The basic mathematical theory underlying the framework is formulated in [5] based on [4]. ORIP works under two fundamental constraints: *i)* all inputs must be integers; *ii)* the image processing task must be linear. The first constraint can be alleviated by using a fixed-point approximation (FXP) for non-integer inputs, with appropriate scaling at the end of the operation [5].

3. Framework Features

ORIP is written in C++ and makefiles are currently available for the Microsoft Visual C++ 2008 (or newer) and the GCC4.0.0 (or newer) compiler. In order to allow for comparisons with non-incremental (conventional) execution, the algorithms provided within ORIP are also implemented in the conventional (non-incremental) manner and are used for comparison purposes. A large set of features are supported via the command line; below we highlight the five most important ones:

- **Number of bitplanes `(-b X)`**: This will allow for the usage of the `X` most-significant bitplanes of the input in the calculation (for both conventional and incremental processing). This parameter impacts the final result of the calculation because it stops the processing after the `X` MSBs have been processed.
- **Bitplane scanning pattern `(-p X,...,Y)`**: this affects only the incremental approach. This parameter defines the way the framework uses bitplanes during the calculation. For example, `-p 2,2,2,2` will read (and process) two bitplanes at a time and provide for terminating bitplanes $n = \{6,4,2,0\}$. Every type of scanning pattern is admissible and the framework checks it according to the bits per sample (in our case, 8 bits per sample are used for greyscale input images).
- **Fraction of completion of last increment `(-l X)`**: this affects only the incremental approach. This parameter selects the coverage fraction of the last step in the scanning pattern. This option takes a decimal value between 0.0 and 1.0 and performs packing, processing and unpacking only for the `X`·100% of the input during the last step in the scanning pattern.

- **Type of kernel `(-t X)`**: for kernel-based calculation (transform decomposition, convolution, or cross correlation), the user can select from a random kernel (with no limit in size) or a small number of predefined kernels (e.g. Gaussian kernels for convolution or the AVC H.264 4x4 transform for block decomposition). We plan to enable the definition of arbitrary kernels from a text file in the near future.
- **Scheduling parameters `(-c X and –j Y)`**: When ORIP is compiled with OpenMP support [7], concurrent execution with a scheduler thread can be performed. The first parameter **`-c X`** is controlling the average time available (in milliseconds) for the processing of each frame. The second parameter controls the timer jitter (in milliseconds) around the average time; this parameter is useful when attempting to simulate the effects of a very dynamic scheduler that changes the termination time imposed for each frame. Once the imposed time per frame is reached, the scheduler thread raises a flag that imposes the immediate termination of the processing of the current frame.

ORIP has several other features of secondary importance, such as the capability to select the dynamic range of the random processing kernel, the block size and search range of motion estimation, etc.

4. Indicative Experimental Results

In order to achieve stable execution-time measurements with high precision in Windows-based and Unix-based platforms, we used the Windows `QueryPerformanceCounter()` function and the UNIX/Linux `gettimeofday()` function and run all programs in highest priority. Only the execution time required for the computation was measured (and converted to milliseconds based on system-specific timing measurement). All I/O time from/to the disk was excluded, since it produced the same overhead for both the conventional and the incremental approaches.

In this paper, we present indicative results for 2D convolution and cross-correlation using three representative CIF video sequences. The block partitioning separates the image into M partially overlapping horizontal "stripes", each of which is the considered to be the input block of samples.

Example 1: The command that performs the convolution with a 12x12 Gaussian kernel whose coefficients are approximated by FXP representation with fractional part set to 8 bits and provides three stop points, at bitplanes $n = \{5,2,0\}$, by using scanning pattern $\{3,3,2\}$:

```
orip_conv_inc_d_v1.exe -i input.yuv -o output.yuv -t 1 -b 8 -p 3,3,2
```

Using a different value for the `–t` parameter, it is possible to choose a different kernel type.

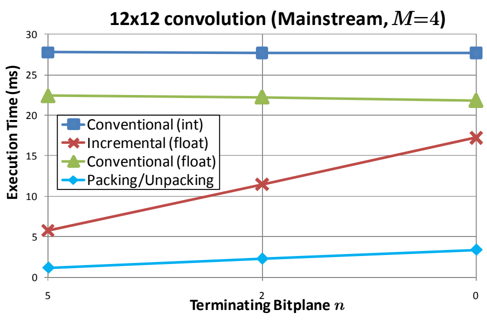
Example 2: The command that performs the cross-correlation with a random 8x8 image block and provides three stop points, at bitplanes $n = \{5,2,0\}$, by using scanning pattern $\{3,3,2\}$:

```
orip_conv_inc_d_v1.exe -i input.yuv -o output.yuv -t 0 -b 8 -p 3,3,2
-m 8 -n 8
```

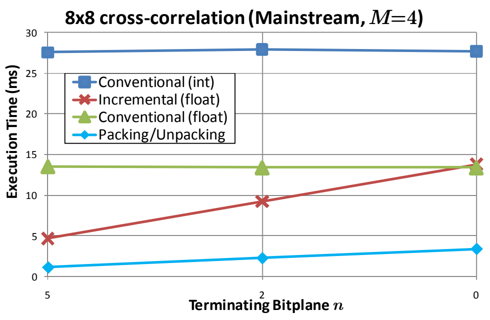
The final summary of the processing provides all the necessary execution time information.

After several runs are performed for each terminating point in a mainstream Dell D630 notebook, it is possible to create a graph of the average processing time for each terminating bitplane. Figure 2(a) shows the results of Example 1. Figure 2(b) shows the results of Example 2. All measurements can be produced by the incremental approach in a single execution, while the conventional approach has to be executed multiple times, each time with different input precision.

In both cases, it was found that $M = 4$ stripes can be packed together without producing any error in the output results. We also measured the corresponding SNR (Table 1) against the result computed at full precision (i.e. terminating bitplane $n = 0$, corresponding to `–b 8`). A visual example of the output for each terminating bitplane for Example 1 is given in Figure 3. The results show that the incremental approach leads to comparable or faster execution than the conventional approach, while it can be arbitrarily terminated and provide the result up to the computed precision. Importantly, unlike the incremental approach, the execution of the conventional approach does not scale down with decreased input precision. Finally, it is interesting to notice that, for the tested platform (mainstream notebook), floating-point representation led to faster execution than integer representation.



(a)



(b)

Figure 2: (a) 2D Convolution result with a 12x12 Gaussian kernel; (b) 8x8 cross-correlation with a random block.

Terminating Bitplane	2D Filtering SNR (dB)	
	12x12 Gaussian	8x8 cross-correlation
$n = 5$	18.88	18.88
$n = 2$	39.35	39.34
$n = 0$	∞	∞

Table 1. Average SNR results for Examples 1 and 2. Both conventional and incremental processing achieve exactly the same results for each terminating bitplane.

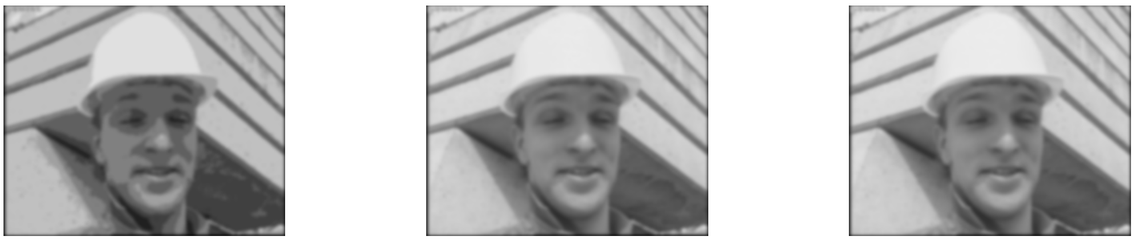


Figure 3. Representative output frame for terminating the computation at $n = \{5, 2, 0\}$ bitplanes (shown from left to right) for the 12x12 Gaussian filtering.

7. Conclusion

Software realizations of computationally-demanding image processing tasks do not currently provide graceful degradation when their clock-cycles budgets are reduced. The proposed ORIP framework performs bitplane-based computation combined with an incremental packing framework in order to realize block transforms, 2D convolution and frame-by-frame block matching. *By construction*, ORIP leads to progressive output-quality improvement when more system resources are utilized. The results with bitplane-based computation indicate that the proposed approach can be comparable or superior to conventional (non incremental) computation for several cases.

References

[1] M. van der Schaar and S. N. Shankar, "Cross-layer wireless multimedia transmission: challenges, principles, and new paradigms," *IEEE Wireless Comm. Mag.*, vol. 12, no. 4, pp. 50-58, Aug. 2005.
[2] Y. Andreopoulos and M. van der Schaar, "Incremental refinement of computation for the discrete wavelet transform," *IEEE Trans. on Signal Process.*, vol. 56, no. 1, pp. 140-157, Jan. 2008.
[3] Y. Andreopoulos and I. Patras, "Incremental refinement of image salient-point detection," *IEEE Trans. on Image Process.*, vol. 17, no. 9, pp. 1685-1699, Sept. 2008.
[4] A. Kadyrov and M. Petrou, "The "Invaders" algorithm: range of values modulation for accelerated correlation," *IEEE Trans. Pattern Anal. Machine Intel.*, vol. 28, no. 11, pp. 1882-1886, Nov. 2006.
[5] D. Anastasia and Y. Andreopoulos, "Software designs of image processing tasks with incremental refinement of computation," *IEEE Workshop on Signal Process. Systems* (SIPS), Tampere, Finland, 2009.
[6] http://www.ee.ucl.ac.uk/~iandreop/ORIP.html
[7] B. Chapman, G. Jost, and R. van der Pas, *Using OpenMP*, MIT Press, 2007, ISBN-13: 978-0-262-53302-7.