Proc. IEEE Workshop on Signal Processing Systems (SIPS), Tampere, Finland, Oct. 7-9, 2009

SOFTWARE DESIGNS OF IMAGE PROCESSING TASKS WITH INCREMENTAL REFINEMENT OF COMPUTATION

Davide Anastasia and Yiannis Andreopoulos

Department of Electronic and Electrical Engineering, University College London, UK

ABSTRACT

We propose software designs that perform incremental computation with monotonic distortion reduction for two-dimensional convolution and frame-by-frame block-matching tasks. In order to reduce the run time of the proposed designs, we combine bitplane-based computation with a packing technique proposed recently. In the case of block matching, we also utilize previously-computed motion vectors to perform localized search when incrementing the precision of the input video frames. The applicability of the proposed approach is demonstrated by execution time measurements on the xo-laptop ("100$ laptop") and on a mainstream laptop; our software is also made available online. In comparison to the conventional (non-incremental) software realization, the proposed approach leads to scalable computation per input frame while producing identical (or comparable) precision for the output results of each operating point. In addition, the execution of the proposed designs can be arbitrarily terminated for each frame with the output being available at the already-computed precision.

Index Terms— *complexity-scalable image processing, incremental refinement of computation, programmable processors*

1. INTRODUCTION

Several popular applications, such as media players, image and video post-processing, and motion estimation and compensation, are being implemented today via software solutions in general-purpose processors. New generations of processors are increasingly powerful and enable more dedicated resource allocation to such real-time multimedia tasks due to multicore designs [1]. At the same time, new generations of software compilers now automatically generate platform-specific optimized assembly code from C++ code [2], thereby enabling platform-independent C++ software solutions to achieve high processor utilization factors.

Existing algorithmic-oriented research focuses on complexity reduction [3]-[5] or complexity scalability for image processing tasks [6]-[8], where computational complexity is decreased and approximate results are produced. Implementation-oriented research focuses on multimedia-driven energy scaling of processors via dynamic voltage scaling [9] [10] in an attempt to provide computational scalability with approximate results. However, previous approaches can only obtain *one operational point* in the complexity-distortion curve [3] [7], without being able to increment the quality of the output with increased computation. In addition, in practical image and video coding systems, complexity does not scale down significantly with decreased source precision (decreased bitrate) [7]. Finally, existing image processing realizations produce an "all or nothing" output: one cannot interrupt the computation when system resources suddenly become unavailable (or when delay constraints are about to be violated) and retrieve a meaningful approximation of the final result[1].

An exception is found in proposals for incremental computation of transforms and salient point detection algorithms [11] [12], where the main principle is: *under a refinement of the source description, the computation of the image processing task refines the previously-computed result.* However, existing work [11] [12] is using arithmetic complexity estimates and no practical realizations are given.

In this paper we address this aspect by proposing a practical software framework for image processing tasks exhibiting incremental refinement of computation. Our software designs of two-dimensional (2D) convolution and block-matching operations combine incremental computation with a recently-proposed packing approach that enables the calculation of multiple limited-dynamic-range integer operations via one 32-bit or 64-bit arithmetic operation. The proposed software designs are validated in two different systems and are also provided online [13].

Section 2 presents the overall framework of incremental refinement of image processing operations in software. Sections 3 and 4 present the proposed algorithms, while Section 5 presents the experimental comparisons and Section 6 concludes the paper.

2. IMAGE PROCESSING TASKS USING INCREMENTAL PACKING AND UNPACKING

A general depiction of the proposed framework for incremental computation based on source refinements is presented in Figure 1. In the following subsection we discuss this framework in more detail, while Subsection 2.2 presents

[1] One could potentially obtain a partial computation of the final result, e.g. parts of the convolution or some block matching results, but not the entirety of the result with graceful degradation.

the basic tradeoffs of the proposed packing approach. Two useful definitions of quantities used in the remainder of the paper are given below.

Definition 1: For any quantity a used in the computation of an algorithm, a^n_{full}, $0 \le n < N$, is the computed value of a when the input consists of bitplanes $N-1$ down to (and including) bitplane n. □

Definition 2: For any quantity a used in the computation of an algorithm, a^n_{bit}, $0 \le n < N$, is the computed value of a when only bitplane n of the input is used. □

The notational conventions of Definition 1 and Definition 2 are extended to matrices[2], e.g. $\mathbf{A}^n_{\text{bit}}$ is the matrix containing the computed coefficients of $\mathbf{A}$ when only bitplane n of the input image is used.

2.1. Overall Framework

As shown in Figure 1, an input image is initially partitioned into M of non-overlapping blocks, whose binary (bitplane-by-bitplane) representation is shown in the middle of the figure, from the most-significant bits (MSBs) to the least-significant bits (LSBs). A total of N *increment layers* are formed by grouping together the bits of all blocks belonging to the same bitplane, where $n = N-1$ corresponds to the MSBs and $n = 0$ corresponds to the LSBs. Hence, one can have maximally $N = 8$ increment layers for an 8-bit input image.

Each increment layer n is also a layer of computation and we calculate the results of M blocks of each layer together using an incremental packing approach. In particular, M blocks $\mathbf{B}^n_{1,\text{bit}},\ldots,\mathbf{B}^n_{M,\text{bit}}$ of one layer are placed together in one block $\mathbf{D}^n_{\text{bit}}$ by:

$$D^n_{\text{bit}}[i,j] = \sum_{m=1}^{M} B^n_{m,\text{bit}}[i,j] 2^{\lambda_{\text{type}}(m-1)\rho}, \tag{1}$$

where $B^n_{m,\text{bit}}[i,j]$ is the (i,j)th value of block $\mathbf{B}^n_{m,\text{bit}}$ ($1 \le m \le M$) that contains parts of increment layer n belonging to the mth spatial block and $\lambda_{\text{type}} = -1$ if 64-bit floating-point arithmetic is used, or $\lambda_{\text{type}} = 1$ if 32-bit unsigned integer arithmetic is used. The last equation shows that the nth bitplane of the mth block is scaled by $2^{\lambda_{\text{type}}(m-1)\rho}$, $\rho > 0$, and is then added to the sum of the previous blocks $1,\ldots,m-1$ of the same increment layer. This leads to a packed increment layer having all M blocks placed on one block $\mathbf{D}^n_{\text{bit}}$ and using integer or floating-point arithmetic. The best choice for the utilized arithmetic is system dependent, as it will be shown by our experiments.

After the packing approach, the desired image processing task op is applied to $\mathbf{D}^n_{\text{bit}}$ for each layer n, $0 \le n < N$, e.g. convolution with kernel $\mathbf{K}$ is performed by:

$$\mathbf{R}^n_{\text{bit}} = \left(\mathbf{D}^n_{\text{bit}} \ op \ \mathbf{K}\right). \tag{2}$$

Depending on the algorithm of interest, one could localize the calculation of (2) around areas of interest based on the previously-computed increment layers (as indicated in Figure 1). This will be used in the block matching task.

If an appropriate coefficient ρ is chosen for (1), it can be shown [14] that the results of all the blocks within increment layer n can be extracted from $\mathbf{R}^n_{\text{bit}}$ if:

- the processing kernel $\mathbf{K}$ contains integers;
- op is a linear operation.

This is based on the so-called "invaders" approach [14], where any integer linear operation can be performed by packing multiple operations together [see (1)], and then unpacking them by the reverse operation performed recursively for all values of all blocks. For floating-point arithmetic ($\lambda_{\text{type}} = -1$):

$$m = 1: \qquad U^n_{1,\text{bit}}[i,j] = \left\lfloor R^n_{\text{bit}}[i,j] + 0.5 \right\rfloor \tag{3}$$

$$\begin{aligned} \forall m \in \{2,M\}: R^n_{\text{bit}}[i,j] &\leftarrow 2^{\rho}\left(R^n_{\text{bit}}[i,j] - U^n_{m-1,\text{bit}}[i,j]\right) \\ U^n_{m,\text{bit}}[i,j] &= \left\lfloor R^n_{\text{bit}}[i,j] + 0.5 \right\rfloor \end{aligned} \tag{4}$$

where $\mathbf{U}^n_{m,\text{bit}}$ is the output increment of the result for block m, $\lfloor a + 0.5 \rfloor$ performs rounding to the nearest integer, and $a \leftarrow f(a)$ assigns $f(a)$ to a. For integer arithmetic ($\lambda_{\text{type}} = 1$), the unpacking is performed by:

$$m = 1: \qquad U^n_{1,\text{bit}}[i,j] = \operatorname{mod}\left(R^n_{\text{bit}}[i,j], 2^{\rho}\right) \tag{5}$$

$$\begin{aligned} \forall m \in \{2,M\}: R^n_{\text{bit}}[i,j] &\leftarrow 2^{-\rho} R^n_{\text{bit}}[i,j] \\ U^n_{m,\text{bit}}[i,j] &= \operatorname{mod}\left(R^n_{\text{bit}}[i,j], 2^{\rho}\right) \end{aligned} \tag{6}$$

where $\operatorname{mod}(a, 2^{\rho}) = a - \left\lfloor a / 2^{\rho} \right\rfloor 2^{\rho}$ is the modulo operation. The selection of the appropriate packing coefficient ρ depends on the specific algorithm being considered. In addition, even though Figure 1 shows all blocks of the input image being packed together, in practice the value of M depends on the dynamic range of the result of each increment layer. These aspects are elaborated further in the following sections.

As shown in Figure 1, after unpacking, the final stage of the proposed computation increments the previously-computed results of increment layers $N-1,\ldots,n+1$ by adding to them the results of the current layer, $\mathbf{U}^n_{1,\text{bit}},\ldots,\mathbf{U}^n_{M,\text{bit}}$:

$$\forall m \in \{1,M\}: \mathbf{U}^n_{m,\text{full}} = \mathbf{U}^{n+1}_{m,\text{full}} + \mathbf{U}^n_{m,\text{bit}}, \tag{7}$$

with $\mathbf{U}^N_{m,\text{full}} \equiv \mathbf{0}$. This leads to computation of the processing task with increasing precision for increasing increment layers, as shown in the visual examples of Figure 1. Due to the utilization of the packing technique, the results of all M blocks are computed *simultaneously* by (2). Depending on the overhead of the packing and unpacking, we expect to save operations in comparison to the direct computation of each layer.

2.2. Discussion

The parameters controlling the proposed approach of Figure 1 are: the total number of increment layers (N), the total number of blocks (M), and parameter ρ that affects the packing of multiple increment layers in one operand $D^n_{\text{bit}}[i,j]$. Ideally, we would like to maximize the packing capability in order to perform as many operations simultaneously as possible [14].

[2] Boldface capital letters indicate matrices; the corresponding italicized letters indicate individual matrix elements, e.g. $\mathbf{A}$ and $A[i,j]$; all indexes are integers; superscripts in matrices or scalars indicate the bitplane number and the frame index, the distinction between the two is identifiable from the context.

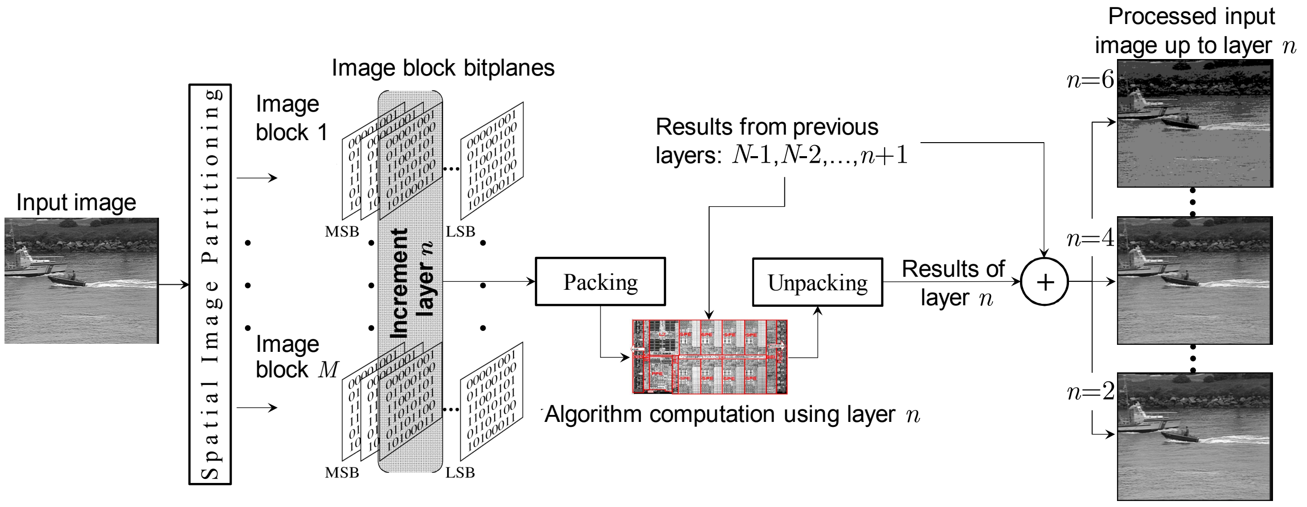
Input image
Spatial Image Partitioning
Image block 1
Image block M
Image block bitplanes
MSB
LSB
Increment layer n
Packing
Results from previous layers: N-1,N-2,...,n+1
Algorithm computation using layer n
Unpacking
Results of layer n
Processed input image up to layer n
n=6
n=4
n=2

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. The computation of each layer can also utilize results from previous layers to reduce complexity.

As analyzed in the original "invaders" algorithm, the packing capability depends on the dynamic range of the operations. Furthermore, if packing with integer arithmetic is desired, ρ has to be integer. The maximum packing coefficient cannot be smaller than 2^{-50} for 64-bit floating-point arithmetic [14], and it cannot be larger than 2^{31} for 32-bit unsigned integer arithmetic, which leads to $\left[M + 0.5\left(\lambda_{\text{type}} - 1\right)\right]\rho = \omega_{\text{type}}$, with $\omega_{\text{type}} = 50$ or $\omega_{\text{type}} = 31$, respectively. If the range of all outputs ($\mathbf{B}^{n}_{m,\text{bit}}$ op $\mathbf{K}$) (for every bitplane n and block m) is contained in the interval $[-A_{\max}, A_{\max}]$, then, for loose packing, $\rho \geq \left\lceil \log_2 A_{\max} \right\rceil + 1$. Selecting the minimum value of ρ satisfying the inequality, we reach

$$M \leq \left\lfloor \frac{\omega_{\text{type}}}{\left\lceil \log_2 A_{\max} \right\rceil + 1} \right\rfloor - 0.5\left(\lambda_{\text{type}} - 1\right). \tag{8}$$

As expected, the number of packed blocks decreases with the increase of the output's dynamic range. The output dynamic range of each layer depends on the algorithm of interest and it will be discussed in the following section. Finally, in order to ensure there is no numerical error in the calculation when packing with floating-point arithmetic, the magnitude of the maximum possible error [14] must allow for correct rounding of the final results, i.e.:

$$2^{-\omega_{\text{type}}} A_{\max} \big/ 2^{-(M-1)\rho} < 0.5\,. \tag{9}$$

In our designs, M is initially derived by (8) and then decreased (if needed) so that (9) holds.

3. INCREMENTAL 2D CONVOLUTION

For an image consisting of $R_{\text{in}} \times C_{\text{in}}$ pixels, the block partitioning of this case separates the image into M partially overlapping horizontal "stripes", each of which is the considered to be the input block of samples, $\mathbf{B}^{0}_{m}$ ($1 \leq m \leq M$), having C_{in} columns. The number of rows in each block is controlled by the input image rows and the packing capability (i.e. the value of M). The convolution filter is given by a $V_{\text{kernel}} \times C_{\text{kernel}}$-coefficient kernel $\mathbf{T}_{\text{conv}}$ and convolution of the m th block is performed by:

$$\forall m \in \{1, M\} : \mathbf{U}^{0}_{m,\text{full}} = \mathbf{B}^{0}_{m,\text{full}} * \mathbf{T}_{\text{conv}}\,. \tag{10}$$

In order to produce the correct result with the block-based calculation of (10), consecutive blocks share a common subset of rows $V_{\text{overlap}} = \left\lfloor V_{\text{kernel}}/2 \right\rfloor$, i.e. the first block ("stripe") is overlapping with the second block vertically by V_{overlap} rows, all subsequent blocks overlap with their previous and next blocks by V_{overlap} rows (above and below the block), and the last block overlaps with its previous block by V_{overlap} rows. When bitplanes of the input are used, the process can be performed for each bitplane n of the m th block by:

$$\forall m \in \{1, M\} : \mathbf{U}^{n}_{m,\text{bit}} = \mathbf{B}^{n}_{m,\text{bit}} * \mathbf{T}_{\text{conv}} \tag{11}$$

and the results are added to the previous ones by (7).

If we consider packing the results in order to accelerate the incremental computation, then $\mathbf{D}^{n}_{\text{bit}}$ is formed by (1) and it is used to compute the packed result of all M blocks by:

$$\mathbf{R}^{n}_{\text{bit}} = \mathbf{D}^{n}_{\text{bit}} * \mathbf{T}_{\text{conv}}\,. \tag{12}$$

The results are unpacked from $\mathbf{R}^{n}_{\text{bit}}$ using (3) and (4) and the final results per bitplane n are derived by (7). Visual examples of Gaussian filtering when $n \in \{6, 4, 2\}$ are given in Figure 1.

The packing capability depends on the worst-case dynamic range $A_{\max}$ of the calculation of one increment layer. This can be pre-calculated when the convolution kernel is known to the system by inserting as input the "worst-case" sub-block:

$$\begin{matrix} 0 \leq i < V_{\text{kernel}} \\ 0 \leq j < C_{\text{kernel}} \end{matrix} : B^{n}_{\max,q}[i,j] = \begin{cases} 1, \text{if } (-1)^{q} T_{\text{conv}}[i,j] > 0 \\ 0, \text{if } (-1)^{q} T_{\text{conv}}[i,j] \leq 0 \end{cases}. \tag{13}$$

for $q = \{0,1\}$ and then:

$$A_{\max} = \max_{\forall q} \left\{ \left| \sum_{i=0}^{V_{\text{kernel}}-1} \sum_{j=0}^{C_{\text{kernel}}-1} B^{n}_{\max,q}[i,j] \cdot T_{\text{conv}}[i,j] \right| \right\}.$$

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 by convolution with an integer kernel followed by inverse scaling after the termination of the calculation and can be accommodated by our framework. Finally, when using packing with integer arithmetic, the incremental approach presented in this section only covers the use of convolution kernels with non-negative coefficients. Some additional calculations are required in order to preserve the sign information correctly when packing with integer arithmetic (this is not an issue with floating-point [14]). This will be detailed in future work.

4. INCREMENTAL BLOCK MATCHING

The problem of block matching between two successive images $\mathbf{I}_{\text{full}}^{0,t-1}$ and $\mathbf{I}_{\text{full}}^{0,t}$ (of $R_{\text{in}} \times C_{\text{in}}$ pixels) can be abstracted as follows. Given the mth non-overlapping block of $C \times C$ pixels in $\mathbf{I}_{\text{full}}^{0,t}$ ($1 \le m \le M$) and a search area of $2W \times 2W$ overlapping blocks in $\mathbf{I}_{\text{full}}^{0,t-1}$, find the $C \times C$ block in the search area that is closest to the mth block. Conventional search algorithms are using non-linear distance criteria, such as the mean-square error or the sum of absolute differences [5]. Hence, they cannot be performed based on the proposed packing framework. However, previous research [4] [15] has shown that approximations of the highly-complex SAD-based full-search motion estimation using simpler bitwise criteria can derive block matching schemes with comparable matching accuracy but with much lower complexity. Since a bitwise distance criterion is a natural fit for the proposed bitplane-based computation, in this paper we focus on *incremental approximations* of full-search block matching under such frameworks. In particular, we consider here the popular one-bit motion estimation of Natarajan et al [4], where a binarization of the input image is performed prior to block matching and the exclusive-OR (XOR) function is used as a matching criterion.

This block matching method starts with the application of an integer high-pass 2D mask $\mathbf{T}_{\text{high}}$ to the input images:

$$\forall \tau \in \{t-1, t\} : \mathbf{R}_{\text{high,full}}^{0,\tau} = \mathbf{I}_{\text{full}}^{0,\tau} * \mathbf{T}_{\text{high}}\,, \tag{14}$$

where we use the 19x19 mask proposed by Erturk [15]:

$$T_{\text{high}}[i,j] = \begin{cases} 1/16, \text{if } (i,j) \in \{(0,9),(3,6),(3,12),(6,3), \\ \qquad (6,9),(6,15),(9,0),(9,6), \\ \qquad (9,12),(9,18),(12,3),(12,9), \\ \qquad (12,15),(15,6),(15,12),(18,9)\} \\ 0, \qquad \text{otherwise} \end{cases}.$$

After filtering, the one-bit representation of the filtered image is formed by thresholding the high-frequency images [4]:

$$\begin{matrix} 0 \le i < R_{\text{in}} \\ 0 \le j < C_{\text{in}} \end{matrix} : R_{\text{bin,full}}^{0,\tau}[i,j] = \begin{cases} 1, \text{if } R_{\text{high,full}}^{0,\tau}[i,j] \ge I_{\text{full}}^{0,\tau}[i.j] \\ 0, \qquad \text{otherwise} \end{cases} \tag{15}$$

Then, block matching is performed between all M non-overlapping blocks of $C \times C$ (binarized) pixels in $\mathbf{R}_{\text{bin,full}}^{0,t}$ ($1 \le m \le M$) and their corresponding (binarized) search areas in $\mathbf{R}_{\text{bin,full}}^{0,t-1}$. For each block m at position (i_m, j_m) within image $\mathbf{R}_{\text{bin,full}}^{0,t}$, this derives the optimal location $\{x_{m,\text{full}}^{0,*}, y_{m,\text{full}}^{0,*}\}$ of the matching block within its search area ($-W \le x,y < W$) in frame $\mathbf{R}_{\text{bin,full}}^{0,t-1}$ by:

$$\{x_{m,\text{full}}^{0,*}, y_{m,\text{full}}^{0,*}\} = \arg\min_{\forall x,y} \sum_{i=0}^{C-1}\sum_{j=0}^{C-1} \Big\{ R_{\text{bin,full}}^{0,t}[i_m+i, j_m+j] \oplus R_{\text{bin,full}}^{0,t-1}[i_m+i+x, j_m+j+y] \Big\} \tag{16}$$

The distance function of (16) is simply the summation of the result of the XOR operation between the current block and each block of the search area, which is the Hamming weight. This can be parallelized by packing 32 values of $\mathbf{R}_{\text{bin,full}}^{0,t}$ and $\mathbf{R}_{\text{bin,full}}^{0,t-1}$ in two unsigned 32-bit operands and using a specific processor instruction or a few low-cost operations for the calculation of their Hamming weight [4].

When all bitplanes $n \in \{N-1, N-2, \ldots, 0\}$ of the input images $\mathbf{I}_{\text{full}}^{0,\tau}$ are processed independently, the first part can be performed by the incremental convolution approach of the previous section. Once the results $\mathbf{R}_{\text{high,full}}^{n,\tau}$ of the convolution have been produced, the binarization of (15) is applied in order to produce $\mathbf{R}_{\text{bin,full}}^{n,\tau}$. Then, for every n, the best match per block is found by (16) using $\mathbf{R}_{\text{bin,full}}^{n,\tau}$ and 32-bit integer packing.

However, the above technique is expected to increase the execution time for the incremental block matching process, as each increment layer applies the full search algorithm. In order to accelerate this case, we utilize the knowledge of the best match found for each block during the previous increment layers $N-1, \ldots, n+1$. This is performed as follows. For the first increment layer $n = N-1$, we perform a fast search using logarithmic-step search [5]. Subsequently, for each block m we only search in the neighborhood of the previously-found best match for this block. The localized search pattern is a spiral search with a fixed distance limit of W_{spiral} pixels horizontally and vertically (see [13] for more details). The use of log-search and the localization of the search around the previously-found best match will produce approximate results per increment layer. Comparisons against the non-incremental full search algorithm in terms of prediction quality (via block matching) vs. complexity are shown in the next section.

5. EXPERIMENTAL RESULTS

For our experiments we used the xo-laptop of the OLPC foundation running its native Linux operating system (denoted as "low-end" profile) and a Dell Latitude D630 mainstream laptop with an Intel Core 2 Duo processor (at 2.2GHz and with 2Gb RAM) running Microsoft Windows XP (denoted as "mainstream" profile). All programs were written in C++ and compiled with the gcc4.1.2 compiler in Linux and with the Microsoft Visual Studio 2008 compiler in Windows, with all default optimizations of "-o2 (maximize speed)" mode in both cases. To achieve stable execution-time measurements with high precision in both platforms, we used the Windows `QueryPerformanceCounter()` function and the 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). We excluded all I/O time from/to the disk, since it produced fixed overhead.

We utilized three CIF video sequences of 300 frames each. For the "low-end" profile, we downsampled the sequences to QCIF format at 10Hz in order to achieve real-time (or near real-time) processing with the xo-laptop. For each task, the results present either the signal-to-noise ratio (SNR) or the peak-signal-to-noise ratio (PSNR) for the CIF sequences (similar SNR/PSNR results were obtained for the QCIF sequences). SNR was measured for all convolution experiments using as reference (noise-free) the result when processing up to the LSBs of each frame ($n = 0$). PSNR was measured for the block matching experiment by using the prediction error of frame-by-frame motion compensation (using the original frames) with the produced motion vectors of each algorithm.

5.1. Incremental 2D Convolution Experiments

We present results with 12x12 and 6x6 Gaussian kernels with their coefficients approximated by FXP representation with fractional part set to 8 bits and 6 bits, respectively. The small kernel is applied in the "low-end" profile and the large one in the "mainstream" profile. We also performed an experiment of block cross-correlation using random image blocks of 8x8 and 4x4 pixels as kernel $\mathbf{T}_{\text{conv}}$ for the two profiles. Indicative results are shown in Figure 2 and Figure 3, where we also report the number of packed blocks (M) achieved by the incremental approach. Results for the conventional approach are shown when using floating-point and integer arithmetic. In this way we demonstrate that the "low-end" profile has better performance with 32-bit unsigned integers, while the "mainstream" profile is faster with 64-bit floating-point arithmetic. For the results of the incremental approach, instead of inserting each bitplane separately in the incremental computation, we inserted pairs of bitplanes together. Per video frame, this provides four termination points for the algorithm's execution, which are indicated by the terminating bitplanes of the figures. The conventional (non-incremental) approach was executed four times, each time using the source precision indicated by the terminating bitplanes in the figures.

The experiments of Figure 2 and Figure 3 indicate that, unlike the conventional approach, incremental computation can achieve extremely scalable complexity with varying source precision. Identical SNR results were obtained for both conventional and incremental algorithms. Importantly, the incremental approach can produce *all execution-time vs. distortion measurements via one single execution*. In other words, if, for any frame, the computation is terminated arbitrarily at a given point by a task scheduler, the results based on the already computed bitplanes of that frame are readily available in the program's allocated memory.

When computation is performed for all bitplanes ($n = 0$), the incremental approach is less efficient in comparison to the best case for the conventional approach. Our experiments demonstrate that the overhead is minimized when the number of terminating bitplanes is smaller or equal to the number of packed blocks M ; this is achieved in the "mainstream" profile. If more terminating bitplanes than M are requested, then the proposed approach becomes less efficient at full precision.

5.2. Incremental Block Matching Experiments

Indicative results of the block matching algorithms are shown in Figure 4. We present the case of $C = W = 16$. Since the convolution kernel $\mathbf{T}_{\text{high}}$ of Erturk [15] only requires 16 additions, it was found experimentally that in the incremental algorithm can perform the convolution directly per input set of bitplanes (rather than use the packing approach) without significant overhead. Similar to the previous case, instead of always inserting individual bitplanes, we inserted the last bitplanes together (as indicated by the terminating bitplanes of Figure 4). This occurred because the results demonstrate that the prediction quality increases marginally when $n < 5$.

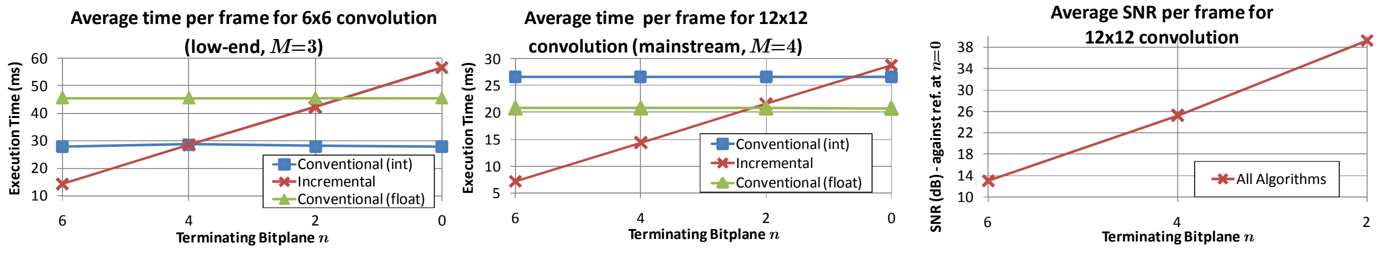

Figure 2. Execution times for FXP convolution and corresponding SNR per CIF frame in function of terminating bitplanes.

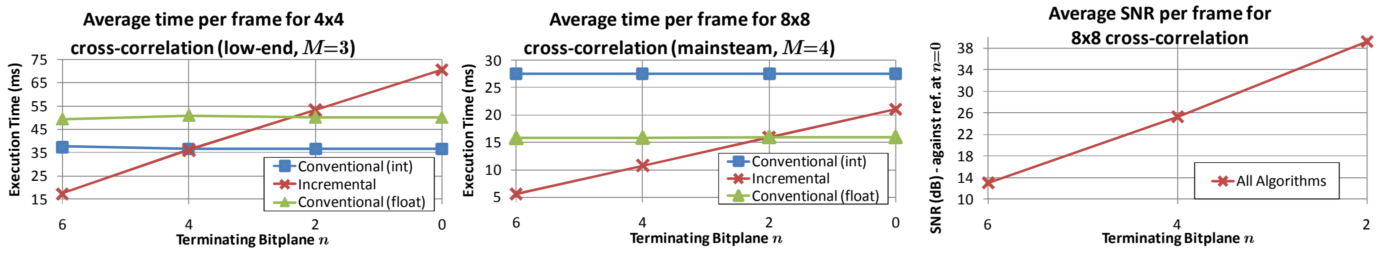

Figure 3. Execution times for cross-correlation and corresponding SNR per CIF frame in function of terminating bitplanes.

The experiments demonstrate that the log-search performed for the first terminating biplane ($n = 7$) provides significantly inferior prediction result for the incremental method as compared to the conventional approach that performs full search. However, the prediction quality of the incremental algorithm becomes comparable to the conventional approach once more bitplanes are processed and the spiral search refines the motion vector per block (0.6dB inferior at $n = 0$). Since the performance seems to saturate when $n < 5$, this can be exploited by the proposed approach to terminate the computation earlier and achieve near real-time performance for both profiles, something that the conventional approach cannot take advantage of, since its execution time does not scale down significantly with decreased source precision.

Our approach presents *successively-refined precision of block matching with additional computation as more bitplanes are processed*, without the need to re-compute the result for each new bitplane. This enables the arbitrary termination of block matching per frame when delay constraints are met (or when resources suddenly become unavailable) and retaining the vectors of all blocks with the already-computed precision.

6. CONCLUSION

A novel, software-based, incremental computation approach for 2D convolution and block matching algorithms was proposed and made available online [13]. The measured run-time of the proposed designs decreases five-fold (or more) with decreased input/output SNR or PSNR, by utilizing a bitplane-based packing technique and reusing previously-computed results. The conventional computation does not achieve such quality vs. complexity scalability, as shown by experimental results on two different platforms. Our method also allows for early termination with graceful degradation and has these features without platform-specific customization. Well-known techniques, e.g. multithreading, can be applied to our approach to improve its execution time without affecting its scalability.

7. ACKNOWLEDGEMENT

This work was supported by EPSRC, grant EP/F020015/1.

REFERENCES

[1] D. Geer, "Chip makers turn to multicore processors," *IEEE Computer*, vol. 38, no. 5, pp. 11-13, May 2005.

[2] A. J. C. Bik, D. L. Kreitzer, and X. Tian, "A case study on compiler optimizations for the Intel Core 2 Duo processor," *Int. J. Parallel Prog.*, vol. 36, Apr. 2008.

[3] V. K. Goyal, and M. Vetterli, "Computation-distortion characteristics of block transform coding," *Proc. IEEE Int. Conf. on Accoust., Speech, and Signal Proc.*, vol. 4, pp. 2729-2732, April 1997.

[4] B. Natarajan, *et al*, "Low-complexity block-based motion estimation via one-bit transforms," *IEEE Trans. Circ. and Syst. Video Technol.*, vol. 7, no. 4, Aug. 1997.

[5] B. Zeng, *et al*, "Optimization of fast block motion estimation algorithms," *IEEE Trans. Circ. and Syst. Video Technol.*, vol. 7, no. 6, Dec. 1997.

[6] K. Lengwehasatit and A. Ortega, "Scalable variable complexity approximate forward DCT," *IEEE Trans. Circ. and Syst. Video Technol.*, vol. 14, no. 11, pp. 1236-1248, Nov. 2004.

[7] D. Turaga, M. van der Schaar, and B. Pesquet-Popescu, "Complexity scalable motion compensated wavelet video encoding," *IEEE Trans. Circ. Syst. Video Technol.*, vol. 15, no. 8, Aug. 2005.

[8] S. H. Nawab, A. V. Oppenheim, A. Chandrakasan, J. Winograd, and J. T. Ludwig, "Approximate Signal Processing," *J. of VLSI Signal Process.*, vol. 15, no. pp. 177-200, 1997.

[9] W. Yuan and K. Nahrstedt, "Practical voltage scaling for mobile multimedia devices," *ACM Internat. Conf. Multimedia*, pp. 924-931, 2004.

[10] E. Akyol and M. van der Schaar, "Complexity model based proactive dynamic voltage scaling for video decoding systems," *IEEE Trans. on Multimedia*, vol. 9, no. 7, pp. 1475-1492, Nov. 2007.

[11] J. Winograd and S. H. Nawab, "Incremental refinement of DFT and STFT approximations," *IEEE Signal Process. Letters*, vol. 2, no. 2, pp. 25-27, Feb. 1995.

[12] 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.

[13] http://www.ee.ucl.ac.uk/~iandreop/ORIP.html

[14] A. Kadyrov and M. Petrou, "The "Invaders" algorithm: range of values modulation for accelerated correlation," *IEEE Trans. PAMI*, vol. 28, no. 11, pp. 1882-1886, Nov. 2006.

[15] S. Erturk, "Multiplication-free one-bit transform for low-complexity block-based motion estimation," *IEEE Signal Process Letters*, vol. 14, no. 2, Feb. 2007.

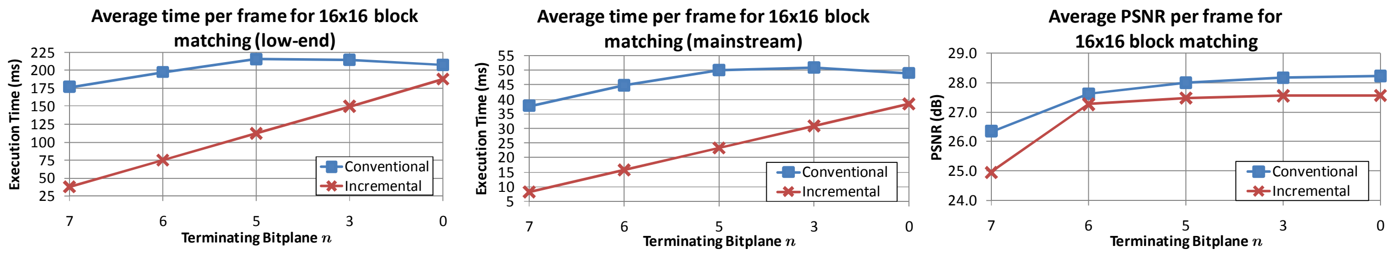

Figure 4. Execution times for block matching and average PSNR per predicted CIF frame in function of terminating bitplanes.