Linear Image Processing Operations With Operational Tight Packing

Davide Anastasia and Yiannis Andreopoulos, *Member, IEEE*

***Abstract*—Computer hardware with native support for large-bitwidth operations can be used for the concurrent calculation of multiple independent linear image processing operations when these operations map integers to integers. This is achieved by packing multiple input samples in one large-bitwidth number, performing a single operation with that number and unpacking the results. We propose an operational framework for tight packing, i.e., achieve the maximum packing possible by a certain implementation. We validate our framework on floating-point units natively supported in mainstream programmable processors. For image processing tasks where operational tight packing leads to increased packing in comparison to previously-known operational packing, the processing throughput is increased by up to 25%.**

***Index Terms*—Accelerated image processing, computation, programmable processors.**

I. Introduction

PACKED linear image processing hinges on the idea that the dynamic range of a 32-bit or 64-bit numerical representation can be used for the concurrent calculation of multiple small-dynamic-range integer operations if the operands are positioned (or "packed") in such numerical representation with appropriate spacing from each other [1], [2]. This has been proposed for a variety of image processing operations such as bound estimation, image cross-correlation and orientation correlation [1], incremental image convolution and motion estimation [2], integer block-transform decomposition [3] and integer wavelet transforms [4].

Consider a linear operation *op* that can be applied to M image blocks $\mathbf{B}_m$ concurrently[1] ($m \in \{0,\ldots,M-1\}, M \geq 2$), using integer operator matrix $\mathbf{K}$:

$$\mathbf{U}_m = (\mathbf{B}_m \,\mathrm{op}\, \mathbf{K}). \tag{1}$$

This can be a block transform decomposition or reconstruction, or a convolution or cross-correlation operation using an integer processing kernel $\mathbf{K}$ [1], [2]. Operational[2] packing first forms a single block $\mathbf{D}$ by

$$\mathbf{D} = \sum_{m=0}^{M-1} \mathbf{B}_m \varepsilon^m \tag{2}$$

with $\varepsilon > 0$ an appropriate packing coefficient. Then, the concurrent processing takes place by

$$\mathbf{R} = (\mathbf{D} \,\mathrm{op}\, \mathbf{K}). \tag{3}$$

Considering the use of an operational real-number representation, such as floating-point, the results can be unpacked sequentially [1]. First, all packed results are shifted to the non-negative region of zero by

$$\mathbf{R}^+ = \mathbf{R} - L_{\min} \cdot \mathbf{J} \tag{4}$$

with $L_{\min} = A_{\min} \sum_{m=0}^{M-1} \varepsilon^m$, $A_{\min}$ the minimum possible value of the results[3] of (1) and $\mathbf{J}$ the unit matrix (matrix of ones). Each result is subsequently unpacked from $\mathbf{R}^+$ by

$$m = 0 : \mathbf{R}^+_{\{0\}} \equiv \mathbf{R}^+, \mathbf{U}^+_0 = \left\lfloor \mathbf{R}^+_{\{0\}} \right\rfloor \tag{5}$$

$$\forall m \in \{1,\ldots,M-1\} : \mathbf{R}^+_{\{m\}} = \frac{1}{\varepsilon}(\mathbf{R}^+_{\{m-1\}} - \mathbf{U}^+_{m-1}), \quad \mathbf{U}^+_m = \left\lfloor \mathbf{R}^+_{\{m\}} \right\rfloor \tag{6}$$

where $\mathbf{R}^+_{\{m\}}$ indicates the contents of $\mathbf{R}^+$ during the mth unpacking and $\lfloor a \rfloor$ the largest integer smaller or equal to a. Finally, the results are derived from $\mathbf{U}^+_0, \ldots, \mathbf{U}^+_{M-1}$ by offsetting to their original range:

$$\forall m \in \{0,\ldots,M-1\} : \mathbf{U}_m = \mathbf{U}^+_m + A_{\min}. \tag{7}$$

The higher the value of M, the higher the execution time reduction offered by operational packing, since more results are calculated concurrently [1], [2].

In this paper we focus on the case of operational packing with real number representations ($0 < \varepsilon < 1$) and in particular with floating-point since i) the parameters for the best-possible packing and unpacking with integer representations are trivial [1]–[4]; ii) unlike integer representations, floating-point representations preserve the sign information for each packed number [1], [2]; iii) programmable processors can offer better native

Manuscript received November 10, 2009; revised December 07, 2009. First published January 26, 2010; current version published February 19, 2010. This work was supported by EPSRC under Grant EP/F020015/1. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Athanassios (Thrasos) Skodras.

The authors are with the Department of Electronic & Electrical Engineering, University College London, London WC1E 7JE, U.K. (e-mail: d.anastasia@ee.ucl.ac.uk; iandreop@ee.ucl.ac.uk.

Digital Object Identifier 10.1109/LSP.2010.2041583

[1]The M blocks can be parts of different images that are processed concurrently, or parts of the same image.

[2]The term *operational* refers to an algorithm or representation realizable by a computer.

[3]The minimum and maximum possible values of the output can be calculated *a-priori* for given *op* and K, under the known dynamic range of the input. An example is given in the following section.

support for floating-point representations in comparison to integer representations thereby enabling higher speed [2]. Hence, the problem we address is: *Given the linear processing algorithm of* (1), *define tight upper bounds for* ε $(0 < \varepsilon < 1)$ and M *that can be used in the operational framework of* (2)–(7).

In their work on tight packing [1], Kadyrov and Petrou propose rules for tight packing, which, under the operational scenario of (2)–(7), are expressed by [(20), [1]]:

$$\varepsilon \leq \frac{1}{A_{\max} - A_{\min} + 1} \text{ and } \varepsilon^{M-1} > 2p \tag{8}$$

with p the maximum numerical error during the packing and processing of (2) and (3), and $A_{\min}, A_{\max}$ the minimum and maximum possible value of the results of (1), respectively. Parameter p is expressed as [1, p. 1883] $p = \max\{|A_{\max}|, |A_{\min}|\}\mu$, i.e., the maximum absolute value produced during the processing, multiplied by parameter μ that represents the relative precision of the computer arithmetic hardware. Since μ stems from the finite precision of the implementation, it can be calculated offline by a simple numerical experiment with the target implementation platform [1]. Even though the proof of [1] shows (by induction) that such rules suffice for mathematically-correct unpacking, it is not shown they are bounds for the operational scenario of (2)–(7). Furthermore, beyond the calculation of μ, no procedure or experiments linking the rules of (8) to a certain operational framework are given in [1]. To the best of our knowledge, only examples of operational loose packing have been presented for $\varepsilon < 1$ [1], [2]. Hence, the problem stated previously remains open.

In this paper, we first derive upper bounds on ε and M (Section II). These bounds turn out to be similar, but not identical, to the rules of (8). Importantly, we demonstrate their tightness via experiments with floating-point representations and derive a procedure for setting the operational parameters for correct execution (Section III). Finally, experimental results comparing tight packing with previously-known loose packing are presented (Section IV), demonstrating for the first time the increase in processing throughput offered by tight packing when higher values for M are obtained.

II. Theory of Operational Tight Packing Revisited

We demonstrate that bounds for ε and M can be derived from the dynamic range of the output, $A_{\text{range}} = A_{\max} - A_{\min}$, that the image processing operation can produce, and from the precision of the operational framework. This range can be calculated if *op* and $\mathbf{K}$ are known. For example, for M blocks of $Q \times Q$ 8-bit unsigned samples (image pixels), $\mathbf{B}_m$, convolved with the $Q \times Q$ kernel $\mathbf{K}$ by $\mathbf{U}_m = \mathbf{B}_m * \mathbf{K}$, we derive $A_{\max}$ and $A_{\min}$ by $(s \in \{0, 1\})$:

$$A_{\text{limit}} = (2^8 - 1) \sum_{i=1}^{Q} \sum_{j=1}^{Q} B_s[i,j] K[i,j] \tag{9}$$

with

$$B_s[i,j] = \begin{cases} 1, & \text{if } (-1)^s K[i,j] > 0 \\ 0, & \text{if } (-1)^s K[i,j] \leq 0 \end{cases} \tag{10}$$

where $A_{\max} = A_{\text{limit}}$ when $s = 0$ and $A_{\min} = A_{\text{limit}}$ when $s = 1$.

Proposition 1: Packing M integers via (2) for linear integer-to-integer processing with output range $A_{\text{range}} = A_{\max} - A_{\min}$, followed by unpacking by (4)–(7), requires:

$$\varepsilon < \frac{1}{A_{\text{range}} + \delta^*} \tag{11}$$

and

$$M \leq \lfloor \log_{\varepsilon}[(A_{\text{range}} + 0.5)\mu] + 1 \rfloor \tag{12}$$

with

$$\delta^* = \arg \min_{\forall \delta \in \mathbb{R}^+} \left\{ \left| (A_{\text{range}} + \delta)^{M-1} (1 - \delta) - A_{\text{range}} \right| \right\} \tag{13}$$

and μ the relative precision of the computer used for the implementation.

Proof of (11): Expanding any element (i, j) of $\mathbf{R}^+$ we have:

$$R[i,j] = (U_0[i,j] - A_{\min}) + \varepsilon (U_1[i,j] - A_{\min}) + \cdots + \varepsilon^{M-1} (U_{M-1}[i,j] - A_{\min}) \tag{14}$$

with $U_m[i,j], m \in \{0, \ldots, M-1\}$, the (i,j) th result for the m th packed block. In order to recover $U_0[i,j]$ correctly via (5):

$$0 \leq \sum_{m=1}^{M-1} \varepsilon^m (U_m[i,j] - A_{\min}) < 1. \tag{15}$$

The upper bound is approached when the linear processing derives $\forall m \in \{1, \ldots, M-1\} : U_m[i,j] = A_{\max}$, i.e., the maximum value for each packed result:

$$\sum_{m=1}^{M-1} \varepsilon^m - \frac{1}{A_{\text{range}}} < 0 \Leftrightarrow -A_{\text{range}} \varepsilon^M + (A_{\text{range}} + 1)\varepsilon - 1 < 0. \tag{16}$$

Furthermore, $\forall m \in \{1, \ldots, M-1\} : U_m[i,j] = A_{\min}$, the lower bound of (15) is achieved, regardless of ε. Hence, the allowed values of ε can be bounded solely based on (16) by

$$\varepsilon \in \left(0, \frac{1}{A_{\text{range}} + \delta^*}\right) \tag{17}$$

with $\delta^* > 0$ derived by the solution of (16) under the marginal condition of equality to zero. The analytic expression deriving the exact value for δ^* from (16) under this marginal condition can be simplified to

$$(A_{\text{range}} + \delta^*)^{M-1} (1 - \delta^*) - A_{\text{range}} = 0. \tag{18}$$

Since the last equation has no closed-form solution for δ^* when $M > 4$, we can express δ^* as the argument minimizing the magnitude of (18), i.e. (13), and use numerical methods (e.g., bisection) to find δ^*. When unpacking any $U_m[i,j], m \in \{2, \ldots, M-1\}$, all admissible solutions for ε have upper bounds that are larger than the one of (17). This is because $\sum_{m=k}^{M-1} \varepsilon^m$ in (16) $(k \geq 1, 0 < \varepsilon < 1)$ is maximized when $k = 1$. As a result, the tightest upper bound for ε, which

ensures all unpackings are mathematically correct, is controlled by the first unpacking. ■

Proof of (12): Assuming (14) under the worst case, i.e. with the maximum value for each element of $\mathbf{R}^+(\forall m \in \{0,\ldots,M-1\} : U_m[i,j] = A_{\max})$ and relative machine precision μ, we have[4]:

$$R[i,j] = A_{\text{range}}\left(\sum_{m=0}^{M-1}\varepsilon^m + \mu\right) + 0.5\mu. \tag{19}$$

In order to recover all $U_m[i,j]$ correctly via (5)–(7), $0 \le m \le M-1$, unpacking via (6) imposes

$$\frac{A_{\text{range}}(\sum_{n=m+1}^{M-1}\varepsilon^n + \mu) + 0.5\mu}{\varepsilon^m} < 1. \tag{20}$$

For the last unpacking, i.e., $m = M-1$, we have $\sum_{n=M}^{M-1}\varepsilon^n \equiv 0$ and, hence, we reach (12) after rounding down to the nearest integer. When any other unpacking $m = M-k, k \in \{2,\ldots,M\}$ is considered, we have $\sum_{n=M-k+1}^{M-1}\varepsilon^n > 0$ and hence we reach bounds for M that are larger or equal to the one of (12). As a result, the tightest upper bound for M is derived by the last unpacking. ■

Remark 1 (Effect of Machine Precision on (11)): The upper bound of (11) did not consider the machine precision. Unlike (12), where the finite precision of the machine (represented by μ) is the reason that makes M a finite number, the upper bound of (11) is imposed by the unpacking process itself and it is valid even under infinite precision. Finite-precision effects will decrease the practical value of ε slightly in some cases, as it will be shown experimentally in Section III. □

Remark 2 (Practical Usage): The practical calculation of the bounds is done as follows.

Step 0 (*Initialization*): Set $L = 1$. Set $M_{\{0\}} \equiv 1, \delta^*_{\{0\}} \equiv \oslash, \varepsilon_{\{0\}} \equiv \oslash$ (the default is no packing capability).

Step 1 (*Increment of packing*): Set $M_{\{L\}} = L + 1$.

Step 2 (*Parameters calculation*): Calculate $\delta^*_{\{L\}}$ from (13) and set $\varepsilon_{\{L\}}$ equal to the bound of (11).

Step 3 (*Packing bound check*): If $M_{\{L\}}$ satisfies (12), increment L by 1 and go to Step 1. Otherwise, the tight packing parameters are settled to $M_{\{L-1\}}, \delta^*_{\{L-1\}}, \varepsilon_{\{L-1\}}$. □

Remark 3 (Link to Prior Work): The results of Proposition 1 are similar to the previously-proposed rules [1] given by (8), but not identical. For example, under non-negative input and kernel values, we have $A_{\min} = 0$, which leads to $M \le \lfloor \log_\varepsilon[(A_{\max}+0.5)\mu] + 1 \rfloor$ under Proposition 1, instead of $M \le \lfloor \log_\varepsilon[2A_{\max}\mu] + 1 \rfloor$ of (8). In general, (8) will approximate the bounds of Proposition 1 only under symmetric dynamic range, i.e., when $A_{\max} = -A_{\min}$. Finally, even though it is proposed in [1] to utilize the remaining space beyond the last packing (i.e., beyond $M_{\{L-1\}}$ from Remark 2) by reducing the range of the last packing, or by the introduction of error in the results of the last block $(\mathbf{U}_{M-1})$, this is not applicable for this work as all packed numbers are under the same dynamic range, $\{A_{\min},\ldots,A_{\max}\}$, since the same operation is performed in all input blocks, and we are only considering error-free operation. □

Next, we validate the derived bounds for practical tight packing in double-precision floating-point representations by presenting the experimental values for ε and M in comparison to the theoretical bounds.

III. Floating-Point Aspects in Tight Packing

It is well known that the mapping of the floating-point representation (with single or double precision) in the IEEE standard is not linear [5]. Floating-point units (FPUs) are designed to have increasingly-finer sampling around zero. Consequently, in an operational environment with an FPU, the experimental value for δ^*, denoted by $\widehat{\delta}$, may be larger than the theoretical estimate of (13) and, hence, the practical value of ε may need to be decreased for correct packing and unpacking in floating-point. This is due to the fact that, under the packing of (2), the working region of each packed sample (i,j) becomes $(U_0[i,j]-0.5, U_0[i,j]+0.5)$, i.e., centers around $U_0[i,j]$ instead of the high-precision region around zero. In the following, we perform a related experiment to demonstrate the practical relevance of the derived bounds and the impact of the precision of FPUs.

We examine the popular case of convolution operations $\mathbf{U}_m = (\mathbf{B}_m \,\text{op}\, \mathbf{K}), m \in \{0,\ldots,M-1\}$, with unsigned 8-bit input samples and non-negative convolution kernels $\mathbf{K}\,(A_{\min} = 0)$ deriving increasing values for $A_{\max}$, which can be calculated by (9) with $s = 0$. We used 25 convolution kernels representing various realistic examples such as: 2–D Gaussian smoothing filters converted to fixed-point (integer) representation [2], image processing kernels (e.g. kernels simulating camera motion effects or smoothing from Matlab's `fspecial()` function), kernels derived from image templates (for template matching via cross-correlation [1], [2]), etc. Our experiments cover: $A_{\max} \in \{10,\ldots,1000\} \times (2^8-1)$. For each kernel, we calculate δ^* by solving (13) with numerical methods (bisection) for each M admissible by (12) and selected the δ^* corresponding to the maximum admissible M. The calculated relative machine precision in the implementation hardware (double-precision floating point realization using an Intel Core Duo 2 processor under Microsoft Visual C++ 9.0) was found to be $\mu = 1.3417592 \times 10^{-16}$ using the test of [[1, eq. (28)]. Per kernel (i.e. per $A_{\max}$ value), the experimental value for δ^* was found by iteratively increasing $\delta^* : \widehat{\delta} = \delta^* + 0.01i, i = 0,1,\ldots,$ until the packed convolution results can be unpacked without error under the operational tight packing framework of (2)–(7). This process has negligible complexity and can be performed at an initialization phase. Furthermore, it is completely realizable in software without requiring access to hardware specifications of a particular system.

Fig. 1 presents the experimental points $(\widehat{\delta})$ versus theoretical prediction (δ^*). The floating-point precision causes slight deviations of $\widehat{\delta}$ from the theoretically-predicted value. Importantly, the solution of (13) predicts the transition point for $\widehat{\delta}$ accurately. Hence, Proposition 1 offers a more precise characterization of the experimental results than the packing rules of (8), which suggest that $\forall A_{\max} : \delta^* = 1$.

Fig. 2 shows the experimentally-derived $\widehat{M}$ versus the theoretical bound of Proposition 1 and the rule of (8) from [1]. We

[4]This includes the term $(A_{\text{range}} + 0.5)\mu$ to account for the maximum possible numerical error, which is upper bounded by the maximum value of the calculation, $(A_{\text{range}} + 0.5)$, scaled by the relative precision μ.

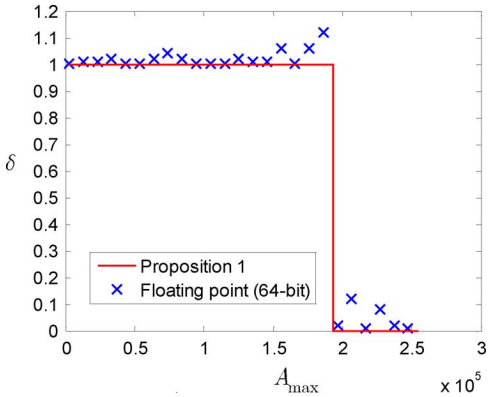

Fig. 1. Graph of $\widehat{\delta}$. (under double-precision floating-point representation) versus δ^* (derived by Proposition 1).

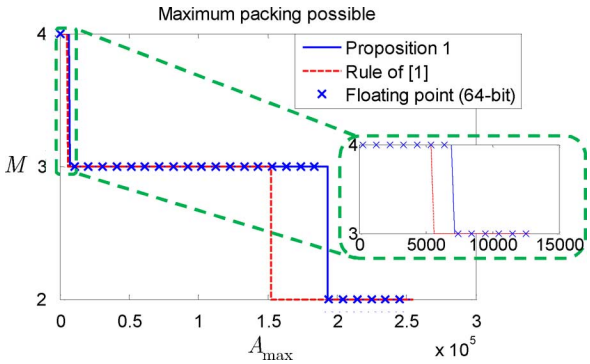

Fig. 2. Experimentally derived $\widehat{M}$ versus its theoretical bound from Proposition 1 and the packing rule of (8) from [1].

have also produced the corresponding graph for $\widehat{\varepsilon}$ versus ε from Proposition 1. For that we measured: $\sum_{\forall\widehat{\varepsilon}}|\varepsilon-\widehat{\varepsilon}| = 2.15\times 10^{-8}$, while for the rule of (8): $\sum_{\forall\widehat{\varepsilon}}|\varepsilon-\widehat{\varepsilon}| = 3.97\times 10^{-8}$, i.e. both theoretical estimates are very close to the measured values. Finally, the bound of (12) predicts the experimentally observed number of packings, $\widehat{M}$, for all cases. Notice that the rule of (8) from [1] does not match the transition points from $M=4$ to $M=3$ and from $M=3$ to $M=2$, since it does not provide the upper bound for the packing capability.

IV. Experimental Results

In order to demonstrate the impact of the proposed tight packing framework in applications, we present example results with two convolution kernels, a 12×12 Gaussian kernel converted to fixed-point assuming nine fractional bits, and a 5×9 integer kernel simulating camera-motion effects derived from `fspecial('motion')` in Matlab. Both kernels derive $A_{\max}$ within $\{1.55,\ldots,1.90\}\times 10^5$, which allows for $\widehat{M}=3$ in the proposed packing, while tight packing via (8) (denoted as "Tight packing [1]") and the operational loose packing environment of [2] (using the entire 8-bit input images), denoted as "Loose packing [2]," achieve $\widehat{M}=2$. All programs where executed via a modification of the ORIP framework [2], [6] to operate using (1)–(7) and, also, using all 8 input bitplanes directly, in order to operate without the incremental-computation features. All programs where compiled with Microsoft Visual C++ 9.0 and utilize the same source code, apart from the packing and unpacking functions that are tailored for each case. They were executed in an Intel Core Duo 2 processor under highest-priority to ensure stable execution time measurements. All I/O time was excluded since it caused the same overhead for all cases. The input content consisted of the luminance frames of several 704×576 YUV progressive video sequences (at 30 frames-per-second). We report the average processing throughput in frames per second for all cases in Table I. Similar results were obtained for several convolution or cross-correlation experiments where an increase of the packing capability was obtained, i.e. in the regions of Fig. 2 that the proposed method offers increased $\widehat{M}$ in comparison to operational tight packing of [1]. The utilized source codes are available online [6].

TABLE I
Throughput Expressed in Frames-Per-Second for Two Convolution Examples. The Proposed Framework Increases the Operational Packing From $M=2$ to $M=3$

Kernel Type	Conventional (no packing)	Packing via [1] [2]	Proposed packing
5×9	11.7	47.5	58.8
12×12	9.2	30.3	36.9

V. Conclusion

We derive theoretical bounds for the spacing between two consecutive operands and the maximum number of operands that can be packed together in a real number representation with certain numerical precision. The experimental validation with linear image processing operations implemented in floating-point hardware indicates the derived bounds are tight. Particular adjustment for the limited cases where the floating-point hardware provides inaccurate results can be done with a very low-complex procedure. As an example of the practical benefits of the theory, a software realization of tight packing demonstrates more than 20% increase in processing throughput for standard-definition progressive video signals in comparison to previously-known tight packing.

References

[1] A. Kadyrov and M. Petrou, "The "invaders" algorithm: Range of values modulation for accelerated correlation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 28, no. 11, pp. 1882–1886, Nov. 2006.
[2] D. Anastasia and Y. Andreopoulos, "Software designs of image processing tasks with incremental refinement of computation," in *IEEE Workshop on Signal Process. Systems*, Tampere, Finland, Oct. 2009.
[3] J. D. Allen, "An approach to fast transform coding in software," *Signal Process.: Image Commun.*, vol. 8, pp. 3–11, 1996.
[4] C. Lin, B. Zhang, and Y. F. Zheng, "Packed integer wavelet transform constructed by lifting scheme," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 10, no. 8, pp. 1496–1501, Dec. 2000.
[5] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Comput. Surv.*, vol. 23, no. 1, pp. 5–47, Mar. 1991.
[6] [Online]. Available: http://www.ee.ucl.ac.uk/~iandreop/ORIP.html