The Fidelity Slider: a User-Defined Method to Trade off Accuracy for Performance in Canny Edge Detector

Valery Kritchallo[1], Erik Vermij[2], Koen Bertels[1], and Zaid Al-Ars[1]

[1]Delft University of Technology, Delft, the Netherlands
[2]IBM Research, the Netherlands

email: v.v.kritchallo@tudelft.nl

***Abstract*— This paper presents the concept of a fidelity slider, which is a user-defined method that enables trading off accuracy for performance in a parallelized application. The slider is defined in the context of the Canny edge detector, but can be generalized to other image processing algorithms. The slider moderates discontinuity issues introduced by an image-slicing technique used to increase the level of the parallelism in the Canny edge algorithm, and allows for strong scalability across multiple cores. The domain decomposition-based technique used by our method is a top-level image-slicing loop incorporated into the algorithm to process segments of an image concurrently. The slider controls three factors to moderate the aggregate output data divergence induced by the parallelized Canny edge algorithm: 1. image slice overlap size, 2. the degree of histogram synchronization, and 3. the edge tracing continuity factor. Results show that the fidelity slider is able to control the tradeoff from a speedup of 7x at 100% accuracy up to a speedup of 19x at 99% accuracy, for an image of 8000x8000 pixels processed on an Intel Xeon platform with 14 cores and 28 hardware threads.**

I. Introduction

The concept of approximate computing is being investigated as a method to increase the performance of algorithms that do not scale very well on a distributed computing infrastructure. Performance can be boosted by relaxing some of the requirements of the algorithm being executed (e.g. data dependencies) to enable a more efficient utilization of the available computational resources. Such methods promise significant improvements in performance at the expense of a lower accuracy of the algorithms [1]–[3].

Image processing algorithms have been popular candidates for applying approximate computing methods to, as they lend themselves readily to parallelization using input data segmentation, an approach known as *domain decomposition* [4]–[6]. However, there has not been a systematic approach to model the degradation effects on the images as a result of approximation, nor has there been a method that enables the user to select a specific level of approximation as a tradeoff for an increase in performance.

This paper presents the design and implementation of a user-controlled *fidelity slider* method that allows the user to control the approximate computing process and to selectively trade off accuracy for performance improvement. We apply the slider approach to the Canny edge detector [7] (referred to as CED further on in the text), a well-known image processing algorithm used in many fields ranging from pattern recognition to computer vision. The approach can however be extended to other image processing algorithms as well, particularly those not easily amenable to efficient parallelization due to their inherent constraints of computational continuity and internal data dependencies, such as e.g. discrete wavelet transform [8].

The concept of user-controlled degradation in image quality has been widely used to enable improvement in the compression ratio of images at the expense of accuracy, in order to increase the efficiency of storage (rather than processing) resource utilization [9]. On the other hand, the majority of image processing approximate computing approaches focus on automatically adjusting the algorithm accuracy based on the available computational resources, rather than through user control [10]–[12].

Among previously published works on parallelization of the CED, [6] is, to our knowledge, the only one that could be meaningfully compared with our in terms of the recorded optimization levels. It reports a speedup of 11 times over sequential achieved on a 16-core CPU for a 2048x2048 pixels test image, but lacks, in our view, sufficient scalability analysis of the presented solution. Furthermore, it doesn't offer any user-controlled scheme of tradeoff between performance and quality of the traced output.

Similarly to the approach used in [6], we improve performance of the CED algorithm by using domain decomposition, a strategy that divides an image into equally-sized segments of pixel rows of the image (or *slices*) and processes them concurrently. Uniquely for our work, the introduced image inaccuracies are then moderated using the fidelity slider that controls a number of correction factors to reduce the aggregate divergence of the parallelized algorithm.

The paper has the following main contributions:

1) we identify three different factors contributing to the error caused by applying approximate computing methods to the CED algorithm;
2) we provide a model to quantify the resulting error;
3) we propose the fidelity slider, which allows the user to control the amount of error tolerated in the resulting image to enable higher performance for the algorithm.

```
/* iterate over image slices */
for (row_ix=0; row_ix<rows; row_ix+=rows_slice) {
#pragma omp task shared(edge_file) if (do_async_tasking)
    {
        /* call the main filter function to process
         * the image slice as a concurrent task */
        canny_par(row_ix, rows_slice, cols, image, ...);
    }
}
```

Fig. 1: The source code fragment implementing the main image-slicing loop, simplified.

TABLE I: Runtime breakdown by functions in the CED algorithm, percents

gaussian_s (Gaussian smoothing)	non_max_s (non-maximum suppression)	apply_hys (hysteresis edge thresholding)	derrivat_x_y (Gaussian derivative x & y)	magnit_x_y (magnitude x & y)	follow_edge (edge tracing)
76.7	8.2	4.4	4.4	3.8	2.3

The rest of the paper is organized as follows. Section II describes the innovative aspects of our CED parallelization. The achieved experimental results are discussed in Section III, followed by Section IV that concludes this paper.

II. Parallelization of Canny edge

A. Canny edge detection algorithm

The Canny edge detector, developed by John F. Canny [7], is a widely known image processing algorithm, description of which can be found in many introductory texts on image processing. Here, we only list the main stages of the algorithm:

1) noise reduction by filtering with a Gaussian-smoothing filter;
2) computing the gradients of an image to highlight regions with high spatial derivatives;
3) relating the edge gradients to directions that can be traced;
4) tracing valid edges using hysteresis thresholding to eliminate breaking up of edge contours.

The baseline sequential version that we used for parallelization, was the CED implementation by Heath et al. [13], [14]. The function and variable names used throughout the text of this section, as well as included in the Figure 2 and Table I, refer to the source code of that implementation.

B. Introduction of the image-slicing loop

The domain decomposition-based strategy we employed to parallelize the sequential version enforces coarse-grained data parallelization onto the application through incorporating a top-level image-slicing loop into its code. In the case of CED implementation, the slices are equally-sized, contiguous blocks of pixel rows that are processed concurrently by asynchronous tasks spawned by a dedicated OpenMP-driven loop (Figure 1). The loop has been parallelized with a single *#pragma omp task* OpenMP directive, which we chose over the more commonly used *#pragma omp for* due to its ability to parallelize non-canonical loops. To host the image-slicing loop, a separate function was added at the top of the program's logic; the (formerly) main function is called from within the loop to process a single slice, instead of the whole image, as before.

The slice size, denoted as the *rows_slice* variable in the source code fragment, is controlled via a parameter in the application's command line that defines number of pixel rows in each slice. If the parameter is not specified, the slice size will be calculated as the number of pixel rows divided by the number of hardware threads supported by the platform.

C. Image-slicing challenges and solutions

Due to the breaks in the computational continuity caused by the introduced image-slicing loop, the following issues have been observed in the edge-traced images rendered by the parallel version:

1) horizontal visual breaks appearing in the image, i.e. blank single-pixel rows between slices, as a result of broken continuity in the Gaussian-smoothing stage of the algorithm;
2) areas in the image rendered differently from the reference output due to the image histogram array no longer computed globally for the entire image, but computed piece-wise within each slice;
3) differently traced edges as a result of violating the logic of the recursive edge-tracing procedure used by the original code that allowed, in principle, for indefinitely long, contiguous edges traced from one arbitrary pixel in the image to another arbitrary one.

To address the above-mentioned issues, we implemented the following additions to our solution.

The parallel code was adjusted such that the image slices included extra overlapping pixel rows blending into the neighbors, in order to mask the visual breaks that resulted from image-slicing. The thereby introduced vertical overlap size (expressed in pixel rows) is an integer parameter varying from 1 to τ_σ, where τ_σ is derived from σ, the input parameter of the Canny edge algorithm that defines the standard deviation of the Gaussian-smoothing filter. The vertical overlap size corresponds to the integer *windowsize* variable found in the sequential code of the filter, and is computed as follows:

$$\tau_\sigma = 2 * \lceil 2.5 * \sigma \rceil \tag{1}$$

For most typical use-cases where σ doesn't exceed 2.0, the value of 10 pixels calculated for τ_σ in accordance with Equation 1 results in a Gaussian-smoothed output identical to that of the sequential version of the algorithm, while the default of 4 pixels provides adequate masking effect. This adjustment fully fixed issue (1), and partially addressed issue (3), with a performance penalty depending on the overlap size and number of slices.

For the produced output to be identical to that of the sequential version, a modification in the parallel code was necessary that would allow using a single histogram of the entire image for all concurrent slice tasks. To implement this, a *histogram synchronization* scheme was developed, where slice tasks performed most of the work independently, and only synchronized with each other briefly (see "histogram sync" block in Figure 2), to compute and share among themselves the single global histogram, before proceeding with edge-tracing within their individual image slices, again concurrently. Based on its own user-defined parameter, the degree of the cross-slice histogram synchronization, the scheme (Figure 2) added an extra degree of control in the accuracy for performance tradeoff within our solution, and partially solved issue (2) of divergence

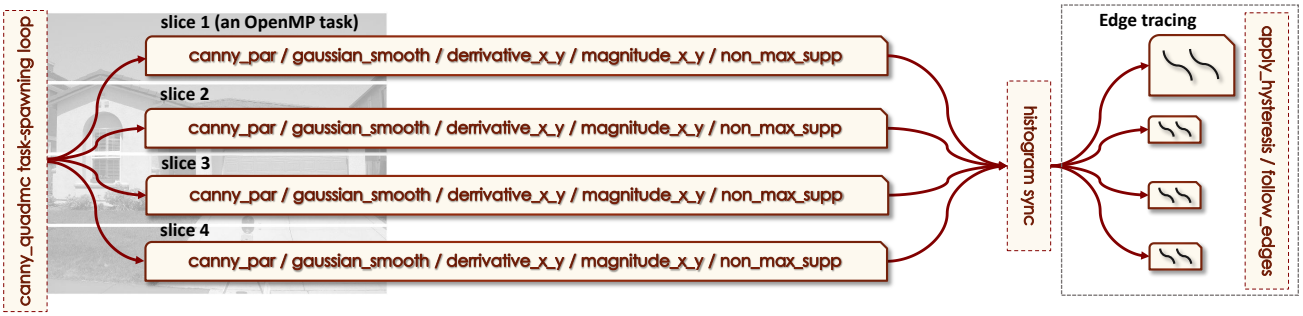

Fig. 2: OpenMP-based parallelization scheme of the CED, drawn as a simplified case of four asynchronous slice-processing tasks entering histogram synchronization before the edge-tracing stage.

with the reference output. The complete solution for issue (2) and (3) could only be found as a part of the fidelity slider design (Section II-E).

D. The fidelity slider: balancing performance and accuracy

In order to fully address issue (2) and (3), a new feature which we call *fidelity slider* has been introduced into the solution. By means of adding elements of approximate computing in a controlled fashion, it allowed to address the principal challenges of the CED parallelization in a more fundamental way.

The first step in implementing the slider is to introduce measurement of the actual binary difference between the parallel version's output and that of the sequential one. For this purpose, we implemented a simple metric expressed as a total sum of average pixel difference between two images (Equation 2), which proved a suitable divergence measure for the purposes of our application, reflecting well the degree of the observable visual difference, as well as more subtle deviations in rendered edges (see Figure 6, explained in more detail in Section III-C.)

The rendering error RE used to calculate the accuracy of the parallel output is computed as the image distance metric described above, that is the average pixel difference between the parallel-produced image p and the sequential image s (both grayscale, the only type of images the CED works with), and is defined as

$$RE(p,s) = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M}\frac{|LPij-LSij|}{255.0}}{N*M} \tag{2}$$

where $LPij$ and $LSij$ are pixel intensity values, and N and M are the images vertical and horizontal dimensions in pixels. The corresponding accuracy percentage value is calculated as

$$AC = 100*(1.0-RE) \tag{3}$$

E. The design of the fidelity slider

The fidelity slider is constructed as a composite parameter driving the strength of the following three factors:

1) the vertical slice overlap size in pixel rows, from 1 to τ_σ. This is the factor introduced to inhibit issue (1); it is denoted as F_1 in Equations 7 and 8 below;
2) the degree of the cross-slice histogram synchronization, expressed as the number of slice tasks synchronized before the algorithm's edge-tracing phase takes place. This is the factor (denoted as F_2 in Equations 9 and 10 below) introduced to inhibit issue (2). This factor progresses from two to all slices synchronized;
3) number of slices rendered in non-concurrent fashion during the last edge-tracing stage, progressing from two to all slices. This is the factor (denoted as F_3 in Equations 11 and 12 below) introduced to inhibit issue (3).

Each of the F_1, F_2 and F_3 factors moderates its own component of the aggregate divergence from the reference image induced by the computational discontinuity issues (1), (2) and (3), respectively, that result from slicing the image. In the equations below, we denote these three divergence components as V_1, V_2 and V_3, correspondingly.

Let integer sv, $1 \leq sv \leq 100$, be the user-defined slider value. Let also integer ns, $1 \leq ns$, be the number of image slices generated by our solution. Given the user-defined slider value sv and the number of image slices ns, we can express the expected accuracy AC_{exp} of the parallel-produced output as a function of the aggregate divergence V_{ag}, in accordance with Equation 3 above:

$$AC_{exp}(sv,ns,\tau_\sigma) = 100*(1.0-V_{ag}(sv,ns,\tau_\sigma)) \tag{4}$$

with V_{ag} constrained such that

$$0 \leq V_{ag}(sv,ns,\tau_\sigma) \leq 1.0 \tag{5}$$

The aggregate divergence V_{ag} is expressed as a sum of the three divergence components:

$$V_{ag}(sv,ns,\tau_\sigma) = V_1(sv,ns,\tau_\sigma) + V_2(sv,ns) + V_3(sv,ns) \tag{6}$$

where V_1, V_2 and V_3 are non-negative floating point values within the same [0, 1.0] interval. Individually, the divergence components can be expressed as follows.

$$V_1(sv,ns,\tau_\sigma) = \sum_{i=2}^{ns} vb_i(F_1(sv,\tau_\sigma)) \tag{7}$$

where F_1 is an integer function of the slider value and the τ_σ parameter calculated as

$$F_1(sv,\tau_\sigma) = \left\lfloor \frac{sv-1.0}{\tau_\sigma} + 1.0 \right\rfloor, \tag{8}$$

τ_σ is an integer derivative of σ, the input parameter to the CED algorithm that defines the standard deviation of the Gaussian-smoothing filter (Equation 1), and $vb_i(F_1(sv,\tau_\sigma))$ is an image-dependent degree of divergence contributed by each

additional slice due to the discontinuity issue (1). The value of vb_i is moderated by F_1 in a manner such that adding a pixel row to the slice overlap along incrementing the factor F_1 results in a steadily decreasing rendering error.

V_2 is expressed as follows:

$$V_2(sv, ns) = \sum_{i=F_2(sv+1,ns)}^{ns} vh_i \tag{9}$$

where F_2 is an integer linear function of the number of slices and the slider value, calculated as

$$F_2(sv, ns) = \left\lceil \frac{sv * ns}{10.0} \right\rceil, \tag{10}$$

and vh_i is an image-dependent degree of divergence contributed by the slices rendered without histogram synchronization (discontinuity issue (2)). Advancing the component's individual slider value from 1% to 100% results in more and more slices using the globally synchronized histogram (as opposed to their locally computed one) and, correspondingly, decreasing rendering error component V_2.

The use of the divisor constant 10.0 in Equation 10 and the ns upper limit in Equation 9 effectively limit the range of distinct fidelity slider values that influence the V_2 component to the [1%, 9%] interval, beyond which the F_2 factor's value results in 100% synchronization across histogram slices and, correspondingly, zero V_2 component error. Imposing such limitation allows to achieve significantly higher performance at a given fidelity value, while keeping accuracy of the output at a sufficient level, than it would have been possible with the factor mapped to the full range of fidelity values between 1% and 100%.

V_3 is expressed as follows:

$$V_3(sv, ns) = \sum_{i=F_3(sv,ns)+1}^{ns} ve_i \tag{11}$$

where F_3 is an integer linear function of the number of slices and the slider value, calculated as

$$F_3(sv, ns) = \left\lceil \frac{sv * ns}{100.0} \right\rceil, \tag{12}$$

and ve_i is an image-dependent degree of divergence contributed by the non-contiguous edges traced within concurrently processed slices (discontinuity issue (3)). Advancing the component's individual slider value from 1% to 100% results in more and more slices jointly edge-traced as a single contiguous fragment of the input image and, correspondingly, decreasing rendering error component V_3. When the F_3 factor's value reaches ns (number of slices), all edges are traced within a single slice identical to the input image, resulting in zero component error.

The composite fidelity slider is exposed as a numeric commandline parameter to the user. Advancing the slider from the fastest / least precise value of 1% to the slowest / 100% precise, results in simultaneous gradual increase of the three F_1, F_2 and F_3 factors strength.

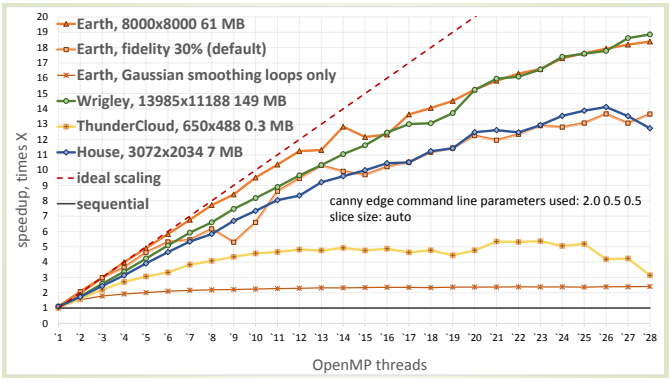

Fig. 3: Benchmark results for the parallelized CED, as registered for four test images at the fidelity level of 1% (unless otherwise noted) with file output disabled. Test platform: Xeon E5-2697 with 14 cores and 28 hardware threads.

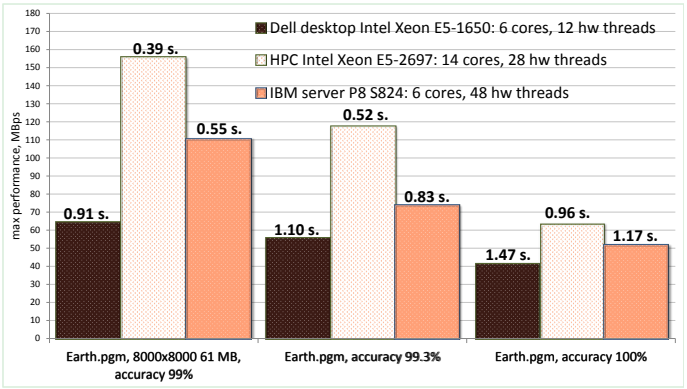

Fig. 4: Maximal parallel performance in MBs per second for the three test platforms at three accuracy levels. Test image: Earth.pgm, 8000x8000 pixels, 61 MB. The figures atop of every bar are the shortest runtime in seconds registered for the platform at each of the three accuracy levels.

III. Results

A. The test and development platforms

In the course of benchmarking our CED implementation, we used the following three platforms. First, an IBM server equipped with two 4.2 GHz POWER8 CPUs both featuring six cores each supporting up to eight threads per core, running Ubuntu kernel v3.16, hereafter called "POWER8". Second, an HPC server with two 2.6 GHz Xeon E5-2697 CPUs, each having 14 cores supporting two threads per core, running Ubuntu kernel v3.13, hereafter called "Xeon E5-2697". And third, a Dell desktop with a 3.2 GHz Xeon E5-1650 CPU, having six cores supporting two threads per core, running Windows 7, hereafter called "Xeon E5-1650". In all our experiments on the dual-socket machines, only a single socket was used. The development platform on the three systems was GCC compiler v4.8.2, v4.9.1, and v4.8.2, respectively, with OpenMP v4.0 as the parallelization environment. The images, code and supplemental material used in this paper are available on the OSF site [15].

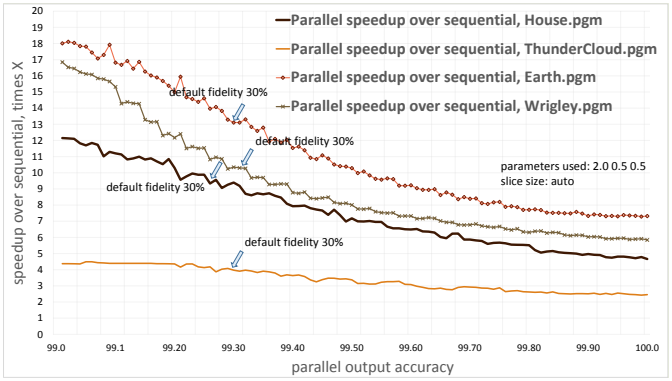

Fig. 5: Impact of the output accuracy on the parallel speedup, as recorded on the Xeon E5-2697 for four test images.

B. Recorded speedups

The maximal parallel execution speedup recorded for this application was 18.66 times over the sequential version, when processing the test 149 MB 13985x11188 pixel Wrigley image on the Xeon E5-2697, with fidelity set at 1% (Figure 3). On the other two evaluation platforms, benchmarking has recorded the highest speedups of 13.33 times over sequential, for the POWER8 (test image: House, seven MB, 3072x2034 pixels), and 7.74 times, for the Xeon E5-1650 (test image: Earth, 61 MB, 8000x8000 pixels).

For the majority of the images we tested (twenty in total), rendering at the 1% fidelity resulted in the accuracy of 98%-99% of the edge-traced output. The chart in Figure 4 displays the highest absolute parallel performance in MBs per second registered for each of the three test platforms at three key accuracy levels (99%, 99.3% and 100%), with the shortest runtime in seconds atop of every bar.

C. Fidelity slider benchmarks

In Figure 5, the benchmark produced results for the four most representative test images are displayed against the vertical speedup axis, with the accuracy value progressing from 99% to 100% along the horizontal one. The speedup is the highest (18.01 times over sequential for our main test image, Earth, 61 MB of 8000x8000 pixel size) at the leftmost position of 99 percent accuracy.

To help in getting an idea of the edge detection rigorousness that can be expected at the lowest fidelity value of 1%, Figure 6 presents variations of a fragment from another our test image, detail-rich Wrigley, with the original picture shown on the left, output of the sequential version in the middle, and that of the parallel one on the right. The difference (which is taking place along a horizontal row at about 1/3 height of the picture where a slice break happened), is rather difficult to discover, unless one is hinted where to look first. The measured accuracy for this rendering was 98.01%.

At the rightmost slider position of 100%, where the performance is the lowest (7.32 times over sequential for Earth), the image difference is zero. In Figure 3, the range of speedups recorded for the default slider value 30% – chosen in our CED implementation for the best combination of speed and quality – can be seen, represented by the curve marked *Earth, fidelity 30% (default)*.

D. Component analysis of the fidelity slider

Figure 7 details breakdown of the aggregate rendering error RE (Equation 2) into its V_1, V_2 and V_3 components (Equations 4 through 12) registered for two test images when edge-tracing them with a fixed number of slices (12). For each of V_1, V_2 and V_3, a separate sequence of benchmarks was performed, with the corresponding F_1, F_2 and F_3 factor's value varying from 1% to 100%, and the two counterpart factors values fixed at their 100% fidelity value. Similarly, Figure 8 illustrates the impact of the F_1, F_2 and F_3 factors (Equations 8, 10 and 12) on the parallel speedup, when varying their individual value and having the two counterpart factors fixed at their 100% fidelity value.

The stair-stepped pattern that the curves exhibit in all four charts in Figures 7 and 8 can be explained by the fact that by definition the range of values allowed for each of the three factors is limited by ns, the number of slices the input image is divided into by the parallelization scheme (twelve, in this benchmark). Furthermore, the [1%, 9%]-interval limitation imposed on the F_2 factor (strength of histogram synchronization, Equation 10) explains the dramatic plunge in the rendering error curves that occurrs between the fidelity values 3% and 7% (Figure 7, Earth), and 5% and 9% (Figure 7, Wrigley), where strengthening of histogram synchronization across four additional slices resulted in a quickly improving quality of the rendering. (Similarly, for the F2-only speedup curves in both charts in Figure 8, which incur a significant drop at the same points.) For the Earth image, advancing further rightwards along the axis, the 30% fidelity value (default) yields the RE aggregate rendering error of 0.000195 (equivalent to the 99.98% accuracy), with the corresponding overall speedup of 6.93 times over sequential.

The two slider benchmark figures allow for at least one important practical observation: with the F_3 factor having such negligible – although still measurable – influence on the aggregate accuracy of the parallel output, it should be possible to achieve substantial performance gains, with only a minor accuracy penalty, by fully relaxing it and keeping the two other factors, F_1 and F_2, at their maximum value. This assumption has been confirmed on the 6-core, 12-hardware thread Xeon E5-1650 platform, where fixing the histogram synchronization level at 100% and the slice overlap size at 10 pixels (which is the maximum τ_σ value computed for the input σ parameter value 2.0), and lowering the edge continuity factor to its minimal level yielded the overall speedup of 7.24 times over sequential, when edge-tracing the Earth test image, with the rendering error measured at only 0.000008. This is compared to 7.74 times and 0.007090, respectively, when rendered at the lowest fidelity slider value 1%, and compared to 5.10 times and 0.0, respectively, when rendered at the highest fidelity slider value 100%, with the same number of slices 12.

IV. Conclusions

In this paper, we demonstrated that a successful application of coarse-grained parallelization and innovative principles of approximate computing through incorporating an image-slicing loop into the CED algorithm allows to achieve highly scalable optimization without using any dedicated hardware acceleration

Fig. 6: A fragment from the Wrigley image (original picture on the left) demonstrating a minor visual difference between the sequential program's traced edge output (middle) and the parallel's one (right), as rendered at the lowest fidelity value of 1%.

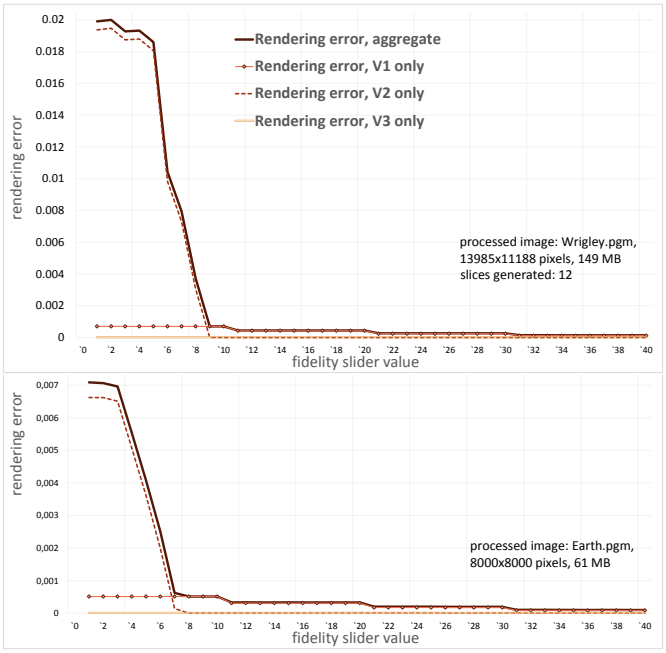

Fig. 7: Fidelity slider benchmark 1: breakdown of the aggregate rendering error into its V1, V2 and V3 components, as recorded on the Xeon E5-1650 for two test images, Wrigley and Earth. Only the part of the slider axis before 40% is shown, as the rendering error converges to zero after that point.

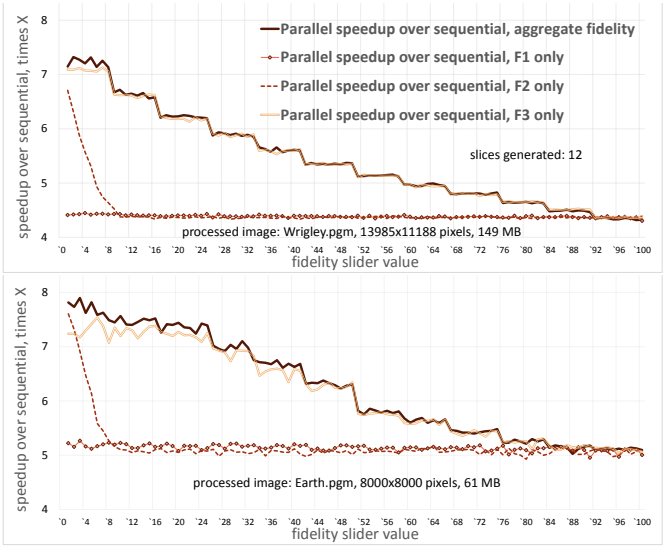

Fig. 8: Fidelity slider benchmark 2: individual impact of the F1, F2 and F3 factors on the parallel speedup when varying the fidelity slider value, as recorded on the Xeon E5-1650 (6 cores and 12 hardware threads) for two test images, Wrigley and Earth.

equipment. The tradeoff between the performance and quality of the output is maintained via the specially-introduced fidelity slider, yielding speedups varying from 18.66x at the accuracy level of 99 percent, down to 7.32x at the accuracy level of 100 percent, as recorded for the fastest benchmark.

REFERENCES

[1] L. Renganarayana *et al.*, "Programming with relaxed synchronization," in *Proc. of the 2012 ACM workshop on Relaxing synchronization for multicore and manycore scalability*. ACM, 2012, pp. 41–50.

[2] M. Rinard, "Parallel synchronization-free approximate data structure construction," in *The 5th USENIX Workshop on Hot Topics in Parallelism*. USENIX, 2013, pp. 1–9.

[3] S. Sidiroglou-Douskos *et al.*, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proc. of the 19th ACM SIGSOFT symposium and the 13th European conf. on Foundations of software engineering*. ACM, 2011, pp. 124–134.

[4] L. H. Lourenço *et al.*, "Efficient implementation of canny edge detection filter for ITK using CUDA," in *13th Symp. on Computer Systems*. IEEE, 2012, pp. 33–40.

[5] A. Z. Brethorst *et al.*, "Performance evaluation of canny edge detection on a tiled multicore architecture," in *Electronic Imaging*. International Society for Optics and Photonics, 2011, pp. 78 720F–78 720F.

[6] L. B. T. Cheikh *et al.*, "Parallelization strategies of the canny edge detector for multi-core cpus and many-core gpus," in *IEEE 10th International Conf. on New Circuits and Systems* . IEEE, 2012, pp. 49–52.

[7] J. Canny, "A computational approach to edge detection," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.

[8] I. K. Park *et al.*, "Design and performance evaluation of image processing algorithms on GPUs," *IEEE Trans. on Parallel and Distributed Systems*, vol. 22, no. 1, pp. 91–104, 2011.

[9] D. Taubman and M. Marcellin, "JPEG2000 Image Compression Fundamentals, Standards and Practice," in *Image Compression Fundamentals, Standards and Practice*. Springer Science & Business Media, 2012, vol. 642.

[10] M. Samadi *et al.*, "Paraprox: Pattern-based approximation for data parallel applications," in *Proc. of the 19th international conf. on Architectural support for programming languages and operating systems*. ACM, 2014, pp. 35–50.

[11] A. Sampson *et al.*, "ACCEPT: A Programmer-Guided Compiler Framework for Practical Approximate Computing," University of Washington, Washington, WA, Tech. Rep., 2014.

[12] J. Ansel *et al.*, "Language and compiler support for autotuning variable-accuracy algorithms," in *Proc. of the 2011 International Symp. on Code Generation and Optimization*, 2011, pp. 85–96.

[13] M. Heath *et al.*, "Comparison of edge detectors: a methodology and initial study," in *IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*. IEEE, 1996, pp. 143–148.

[14] "Edge Detector Comparison," 1996–2015. [Online]. Available: http://marathon.csee.usf.edu/edge/edge_detection.html

[15] "The Canny edge / QuadMC parallelization project," 2015. [Online]. Available: https://osf.io/i725h/