On Quantification of Accuracy Loss in Approximate Computing

Ismail Akturk, Karen Khatamifard, Ulya R. Karpuzcu
University of Minnesota, Twin Cities
{aktur002, khata006, ukarpuzc}@umn.edu *

Abstract

Emerging applications such as R(ecognition), M(ining), and S(ynthesis) suit themselves well to approximate computing due to their intrinsic noise tolerance. RMS applications process massive, yet noisy and redundant data by probabilistic, often iterative, algorithms. Usually the solution space has many more elements than one, rendering a range of application outputs valid, as opposed to a single golden value. A critical step in translating this intrinsic noise tolerance to energy efficiency is quantification of approximation-induced accuracy loss using application-specific metrics. This article covers pitfalls and fallacies in the development and deployment of accuracy metrics.

1. Introduction & Background

In this paper, we refer to *approximate computing* to represent a full bag of tricks – possibly spanning multiple levels of the system stack – which exploit algorithmic noise tolerance for energy efficiency. The bag of tricks encompasses various degrees of approximation to trade off computation accuracy for performance or power: precision reduction [25, 9, 19]; computation perforation [3, 20, 18]; relaxation of execution semantics in the form of approximate consistency [12, 17, 16, 18] or hardware simplification [8, 2, 5, 6, 9]; and embrace of errors [11, 7, 21, 10].

To be able to quantitatively characterize how the complex trade-off space of accuracy vs. energy efficiency evolves under each trick, we need robust metrics to capture approximation-induced loss in computation accuracy. Table 1 depicts widely adopted accuracy metrics classified according to the data type of application outputs, considering example benchmarks from Parsec [1], Splash2 [24], Rodinia [4], Parboil [22], MineBench [15], and Stamp [13] suites.

Classes I and II generate numeric output data in scalar, or multi-dimensional (set of vectors or matrix) formats. To quantify approximation-incurred degradation, recent work deploys variants of the *distortion* metric introduced by Misailovic et al. [14]. *Distortion* is defined as the average, across all output elements, of the *deviation* per numeric output element from its *exact* (non-approximate) value. How the *deviation* per output value is captured gives rise to the variation in distortion-based metrics. For example, if *deviation* per output value is calculated by the *square of the difference between exact and approximate values*, *distortion* falls back to *mean square error*.

Class III encompasses clustering, data mining, searching, and sorting applications which generate compound output data. For example, *ferret* from Parsec conducts content-based similarity search in an image database (Table 1). For each query, a pre-set number of similar images – ranked by similarity to the query image – constitutes the output. To quantify approximation-incurred degradation, recent work relies on metrics based on # mismatches between the *exact* and *approximate* output image lists.

Class IV generates multi-media, i.e. image or video, output. Recent work mostly deploys PSNR (Peak Signal to Noise Ratio) to measure the approximation-induced degradation. However, the alternative metric, SSIM (Structural Similarity Index) is shown to match human perception better than PSNR [23].

2. Fallacies & Pitfalls

In this section, we provide case studies to cover pitfalls and fallacies in the development and deployment of accuracy metrics.

2.1. Validity vs. Accuracy

Under approximation, particularly under high-risk, high-reward techniques such as approximate synchronization, execution may deviate from its *exact* trajectory in various ways, possibly preventing program termination. Even if the program terminates successfully, the output may become excessively corrupt to be considered *valid*. Even worse, accuracy metrics may not always be able to capture such *invalid* execution outcome. This may leave us with accuracy metrics pointing to negligible degradation for *invalid* output data. Application-specific *validity* checks can remedy this problem.

Case Study 1: *dedup* from Parsec implements a file compression algorithm. The *deviation* in the output file size represents the accuracy metric (Table 1). Under approximation, the file size of an *invalid* output – a corrupt file – may even become equal to the file size of the *exact* output. As a validity check, we should try to decompress the output file before deploying accuracy metrics. Depending on the magnitude of the corruption, decompression may not always be possible. Only upon validating the decompressed file against the original, we can start with the accuracy analysis.

Case Study 2: *fluidanimate* from Parsec simulates the physical interactions between a set of fluid particles within a bounded space. The output tabulates positions (along with

*This work was supported in part by the National Science Foundation under grants CCF-1421988, XPS-1438286; and by DARPA under HR0011-12-2-0019.

Class	Output Data Type	Accuracy Metric	Examples		
			Application (Suite)	Domain	Metric
I	Numeric: scalar	Deviation [14] in output value	canneal (Parsec)	Optimization	Dev. in cost
			dedup (Parsec)	Compression	Dev. in file size
II	Numeric: multi-dimensional	Distortion [14] based	fluidanimate (Parsec) barnes, water (Splash2)	n-body simulation	Dist. in "body" positions
			bodytrack (Parsec) particlefilter (Rodinia)	Computer Vision	Dist. in coordinates
			cholesky, lu (Splash2)	Linear Algebra	Dist. in elements
			histo, tpacf (Parboil)	Histogram	
III	Compound	Based on # mismatches Positional error	streamcluster (Parsec) kmeans (Stamp)	Clustering	Based on # mismatches
			UtilityMine (MineBench)	Data mining	
			ferret (Parsec)	Similarity search	
			radix (Splash2)	Sorting	Positional error
IV	Multi-media	Peak Signal to Noise Ratio (PSNR) Structural Similarity Index (SSIM)	raytrace (Splash2) volrend (Splash2)	Computer Vision	PSNR, SSIM
			x264 (Parsec)	Video Encoding	

Table 1: Widely adopted accuracy metrics classified according to the data type of application outputs.

several other physical characteristics such as velocity or acceleration) of the particles at the end of the simulated time frame. Under approximation, the reported positions may exceed the boundaries of the simulated space to render an *invalid* output, however, due to averaging-out effects (or due to relatively small boundary overflows) *distortion* may fail to capture such invalid execution. On top of this, *approximate* execution may drop a subset of the simulated particles to render an incomplete, hence, *invalid*, output. We observed both types of *invalid* outcome under approximate synchronization, where our *distortion* metric (detailed below) reported negligible degradation.

Validity vs. Accuracy: Accuracy metrics may point to negligible degradation even if the *approximate* execution outcome is *invalid*, rendering application-specific *validity* checks critical.

In the following, we confine our discussion to the accuracy analysis of *valid* execution outcome only.

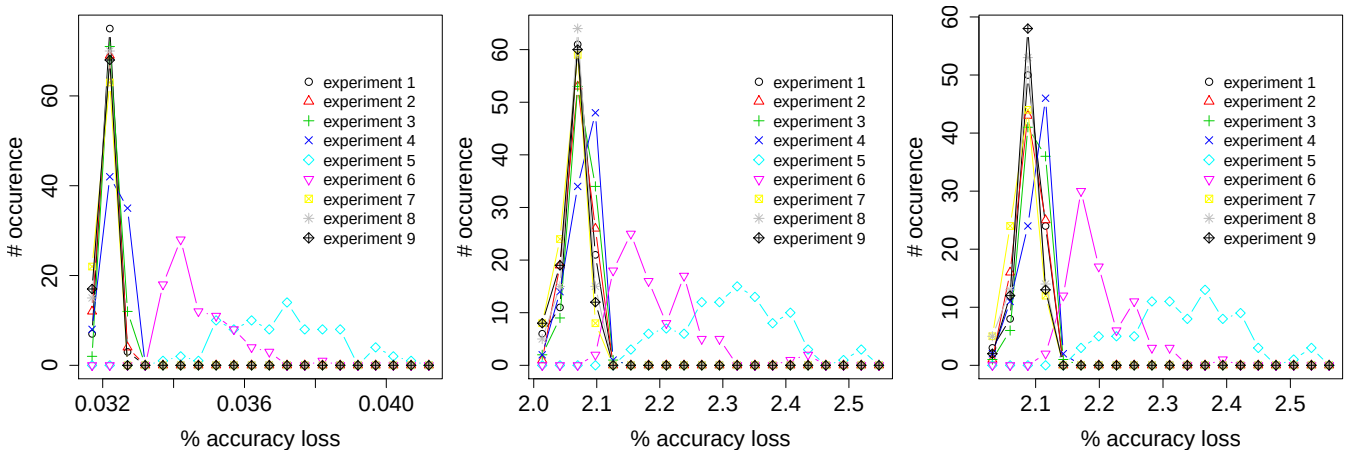
2.2. Absolute vs. Relative Loss in Computation Accuracy

Without loss of generality, we can devise both *absolute* and *relative* accuracy metrics: For example, for Class I from Table 1, we can directly deploy the scalar numeric output value as the *absolute* accuracy metric. This translates into the output file size for *dedup*. By evaluating the very same *absolute* metric under the *exact* execution, we can explore the accuracy vs. energy efficiency trade-off space for a given benchmark and configuration. Oftentimes, however, we are interested in how the trade-off space changes, considering different applications. Relative accuracy metrics can facilitate comparison across multiple applications, by capturing the approximation-incurred degradation *relative* to the *exact* outcome, on a per application basis.

Case Study 3: For n-body simulation (*fluidanimate*), Figure 2 shows the coordinates of four sampled bodies (particles), under *exact* and *approximate* execution, respectively. Under approximation, each body moves in the direction of the arrows. Accordingly, each body moves away by the very same distance from its *exact* position.

Let us measure the *deviation* per body (Section 1) by the *relative displacement in x-axis*, i.e. the ratio of the x-displacement under approximation and the x-coordinate of the *exact* position. This renders a percentage *deviation* of 0.2/0.1=200% for bodies B1 and B3; and of 0.2/0.9=22.3% for bodies B2 and B4, respectively. The measured *deviation* for B1 and B3 is significantly higher when compared to B2 and B4, where each body moved away by the very same distance from its *exact* position! Thus, we failed to capture the actual *deviation*. This observation applies to the symmetrical analysis on the y-axis. For the same x (or y) displacement, we will always measure a larger *deviation*, the closer to the origin the mis-placed body is. This problem persists for the alternative definition for *deviation*, the *relative change in the distance from the origin*, which captures the combined displacement along multiple axes (and possibly accentuates the measurement error). Both definitions rely on coordinates or distances with respect to the origin, and thus, are oblivious to the size or span of the simulated space. The situation gets even worse if the baseline for normalization assumes a very close value to zero. Changing the origin does not help. One way to mitigate this problem is changing the baseline for normalization (i.e. the denominator of the *deviation* metric) to the span (e.g. the longest distance) of the simulated space.

Relative Metrics: The baseline for normalization should not introduce any bias on *deviation* metrics (Section 1). Otherwise, we can easily measure a significantly different *deviation* for the very same actual degradation under approximation.



(a) $M_{distance}$ (b) M_{cell} (c) $M_{cell-distance}$

Figure 1: % accuracy loss of *fluidanimate* under approximate synchronization, considering three different metrics (Section 3).

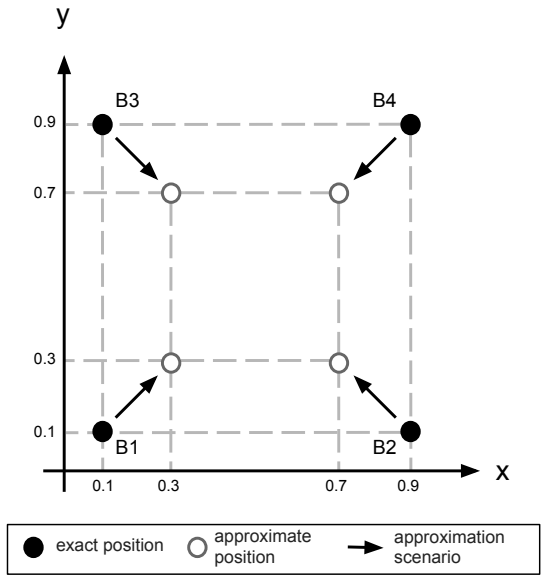


Figure 2: Case study 3, considering n-body simulation.

2.3. Averaging Effects

As depicted in Table 1, most classes of benchmarks generate multi-dimensional numeric output data. In this case, *distortion*-based metrics derive the cumulative degradation under approximation by taking the average of *deviation* (Section 1) across all elements constituting the output. Accordingly, *distortion*-based metrics can easily mask the variation in the *deviation* on a per element basis.

Case Study 4: Let us compare two execution outcomes for *fluidanimate* under approximation: In the first scenario, *all* particles shift by 1% with respect to their expected positions under *exact* execution. In the second scenario, on the other hand, only 1% of the particles move away from their *exact* positions, but by much greater than 1%, such that both scenarios render the very same cumulative degradation of 1% – as calculated by an averaging *distortion* metric. Depending on the execution context, scenarios deemed equivalent by the accuracy metric may associate with very different levels of *acceptability*. To be able to differentiate between such scenarios, we should devise variation-aware accuracy metrics. To introduce variation-awareness to the *distortion*-based metric from the case study, we can augment it with the maximum of per-particle displacement and/or the fraction of displaced particles.

Averaging Effects: *Distortion*-based metrics derive approximation-incurred degradation by taking the average of *deviation* (Section 1) across all elements constituting the output; hence, can easily mask the variation in per-element *deviation*, and report the same accuracy loss for very different variation profiles. However, execution outcomes deemed equivalent by such accuracy metrics can associate with very different levels of *acceptability*. To introduce variation-awareness, accuracy metrics should include extremes of per-element *deviation* and/or the fraction of degraded elements.

2.4. (Non)Determinism under Approximation

High-risk, high-reward approximation techniques to embrace errors or to relax basic execution semantics are of probabilistic nature. So are the application domains which lend themselves well to approximation. These applications may generate different outputs under the very same configuration. The complication stems from not being able to guarantee, under the *exact* execution, that no output exists to deliver a more accurate result than an observed output. To mitigate this, we need to tame the sources of non-determinism: We can enforce single-threaded runs in extracting the outcome under the *exact* execution. Moreover, we can configure the application to deliver the maximum possible output accuracy for a given input.

To this end, for algorithms relying on iterative refinement, we can remove any cap on the maximum number of iterations, or tighten convergence criteria.

At the same time, we need to ensure statistical significance of the estimated accuracy loss under *approximation*. To this end, we can repeat the experiments for a given configuration a preset number of times. Different applications and approximation techniques may give rise to different spectra for the execution outcome. We may observe no program termination, termination with consistently *invalid* outputs, *valid* outputs spanning a wide accuracy loss interval, or some mixture thereof. *Valid* outputs shape the trade-off space of accuracy vs. energy efficiency, however, the existence of non-terminating and *invalid* executions suffice to render an approximation technique infeasible unless safety nets are present. Depending on their complexity, safety nets incur an energy efficiency overhead, which should be reflected to the trade-off space. For example, if we rely on re-execution as a safety net upon encounter with *invalid* or non-terminating cases, we should report the power and performance overhead of re-execution, weighed by the expected frequency of occurrence.

Case Study 5: Figure 1a captures how the percentage accuracy loss in the outputs of *fluidanimate* change under approximate synchronization. Following the guidelines from Section 2.2, the *distortion*-based accuracy metric, $M_{distance}$, captures the *deviation* per particle by the absolute distance between *exact* and *approximate* positions normalized to the maximum possible distance within the simulated space (i.e. the diagonal of the bounding box of all particles). Experiments 1-9 each corresponds to the relaxation (i.e. removal) of a different synchronization point of the application[1]. Each experiment is repeated 100 times. For each experiment, the y-axis depicts the # occurrences (over 100 repetitions) of a % accuracy loss value[2]. We observe that the % accuracy loss does not span a wide interval. However, depending on the context, even such numerically negligible degradation may be barely *acceptable*. This is why it is always safer to report the span of accuracy loss than a single point in the trade-off space. In total, we run 900 experiments with one resulting in non-terminating, 14, in *invalid* execution (Section 2.1). The outcome of these 15 experiments is not visible in Figure 1a.

Case Study 6: In Figure 3, we compare the outcome under approximate synchronization (Experiment 3 from Figure 1a) with the outcome by excluding any approximation. All runs correspond to 16-threaded execution. The points labeled by *no approximation* capture how the output accuracy of the execution (excluding any approximation) varies across 100

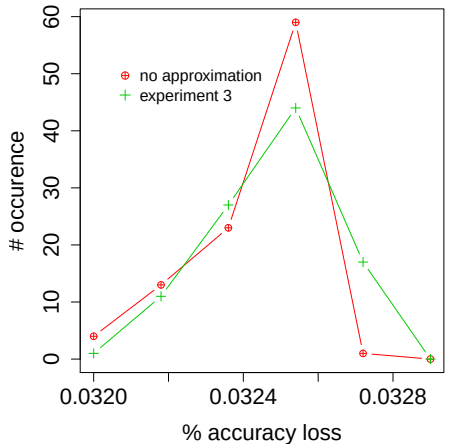
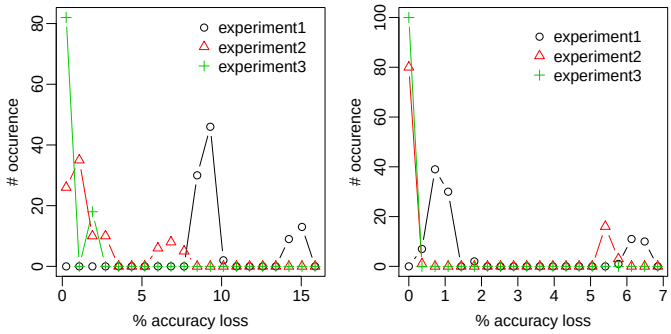


Figure 3: % accuracy loss of *fluidanimate* under approximate synchronization vs. by excluding any approximation, measured by $M_{distance}$ (Section 3).

repetitions due to the inherent non-determinism of *fluidanimate*. We observe that the outcome under *approximation* closely tracks the outcome with any approximation excluded. The inherent non-determinism in the application renders a sizable % accuracy loss even if we exclude approximation. *fluidanimate* divides the simulated space into a grid of cells. Each grid cell encompasses a number of particles. Capturing interactions between the centers of mass of grid cells instead of individual particles reduces algorithmic complexity. The non-determinism under *no approximation* stems from the updates to the border cells: the border cells can be updated by more than one thread concurrently, since the implementation does not enforce any deterministic order. Moreover, these updates involve finite-precision floating point arithmetic which is neither commutative nor associative.



(a) $M_{cluster-assign}$ (b) $M_{cluster-center}$

Figure 4: % accuracy loss of *kmeans* under approximate synchronization for two different metrics (Section 3).

Case Study 7: Figure 4a captures how the % accuracy loss in the output of *kmeans* from Stamp changes under approximate synchronization. *kmeans* relies on iterative refinement in partitioning its input data points into k clusters. Every it-

[1] The synchronization point relaxed resides in line 732 for experiment 1; 741 for 2; 834 for 3; 843 for 4; 1133 for 5; 1135 for 6; 1139 for 7; 1141 for 8; and 1143 for 9, of pthreads.cpp file, respectively.

[2] For this and the following analysis, we run the benchmarks 16-threaded, on a 16-core node comprising 2 eight-core Sandy Bridge E5-2670 processors of 2.6GHz, 32KB instruction and data L1, 256KB shared L2, 20MB LLC, and 64GB memory. Unless otherwise noted, we deploy as input size a simsmall-equivalent [1]. We discuss the sensitivity to the input data (size) in Section 2.5.

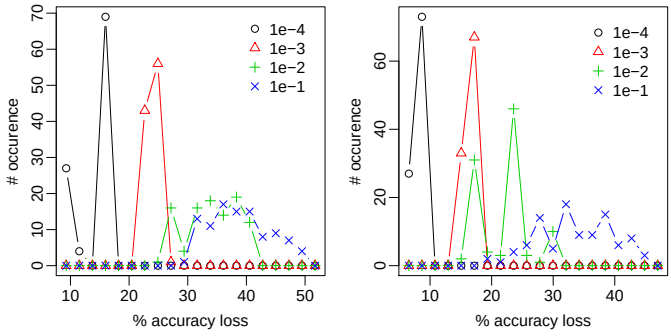
eration, the algorithm assigns each input point to the closest possible cluster (i.e. the cluster of minimum Euclidean distance to the point). Then follows the update of the cluster centers to reflect the new assignments. The algorithm terminates once the assignments stabilize. The accuracy metric, $M_{cluster-assign}$, is the ratio of points which were assigned to a different cluster under *approximation*, when compared to the *exact* outcome. To extract the *exact* assignments, we run the benchmark single-threaded and enforce very strict convergence criteria. Experiments 1-3 each corresponds to the relaxation (i.e. removal) of a different synchronization point of the application: the three critical sections protect the updates to the cluster centers; the accesses to the list of points; and the updates to the variable controlling converge, respectively. We repeat each experiment 100 times. We observe that the % accuracy loss spans a wider interval when compared to *fluidanimate*. However, none of these experiments cause *invalid* or non-terminating execution, as opposed to *fluidanimate*.

Case Study 8: We observe a similar trend if we deploy a different approximation technique than relaxed synchronization: Our *kmeans* implementation relies on iterative refinement guided by Euclidean distances. This time, we assume that the calculation of Euclidean distance between points and cluster centers is mapped to approximate hardware of reduced precision. Figure 5a captures how the % accuracy loss in the output changes, if we stick to the same accuracy metric as in Figure 4a. Each experiment, repeated 100 times, reflects the execution under a specific precision. The legend shows the experiments in the direction of monotonically decreasing precision: each value corresponds to the relative error (in Euclidean distance calculation) incurred due to precision reduction. We observe that the data points tend to move right with a wider spread – to render more experiments with higher loss in output accuracy, as the precision reduces.

(Non)Determinism: In order to tame sources of nondeterminism in characterizing the accuracy of *exact* execution, we need to enforce single-threaded runs, and configure the application to deliver the maximum possible output accuracy for a given input by tightening convergence criteria. At the same time, we need to ensure statistical significance under *approximation*. Different applications and approximation techniques give rise to different spectra for the execution outcome: no program termination, termination with *invalid* outputs, *valid* outputs spanning a wide accuracy interval, or some mixture thereof. *Valid* outputs shape the trade-off space of accuracy vs. energy efficiency, but the existence of *invalid* executions necessitates safety nets. The overhead of safety nets should be reflected to the trade-off space.

2.5. Impact of Input Data

We should also consider different inputs in quantifying the complex trade-off space of accuracy vs. energy efficiency.



(a) $M_{cluster-assign}$ (b) $M_{cluster-center}$

Figure 5: % accuracy loss of *kmeans* under precision reduction for two different metrics (Section 3).

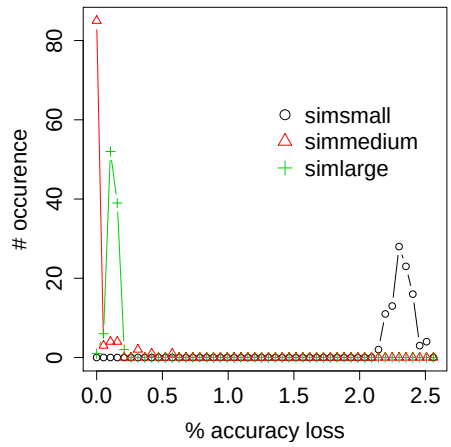


Figure 6: Impact of input size on % accuracy loss for *fluidanimate* under approximate synchronization, measured by metric $M_{cell-distance}$ (Section 3).

The sensitivity of the output accuracy to inputs can vary considerably depending on the approximation technique and the application domain. Inputs may differ both in size and data spread, and both can affect computation complexity and an application's response to approximation.

Case Study 9 (Sensitivity to Input Size): Figure 6 tabulates how the % accuracy loss of *fluidanimate* changes, considering different input sizes. As we move from *simsmall* to *simlarge*, the number of particles changes from 35K to 300K, and the median % accuracy loss across 100 experiments changes notably, but not monotonically.

Case Study 10 (Sensitivity to Input Data): Figure 7 tabulates how the % accuracy loss of *kmeans* changes, considering different input data values. To capture the sensitivity of output accuracy to changes in the distribution of input points, we start with 8K points, with every 1K points having the same coordinates (rendering 8 distinct clusters)[3]. We then add uniform random noise of varying magnitude to all of these 8K

[3] In our *kmeans* implementation, *simsmall* has 2K; *simmedium*, 16K; and *simlarge*, 64K points.

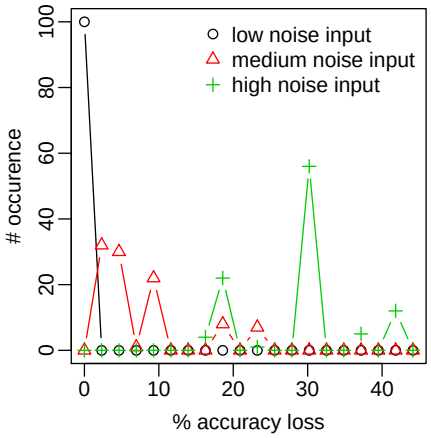


Figure 7: Impact of input data on % accuracy loss for *kmeans* under precision reduction, measured by metric $M_{cluster-assign}$ (Section 3).

coordinates. We quantify the magnitude of noise in three levels; low, medium, and high. We test these inputs under the highest precision reduction from Figure 4a. We repeat each experiment 100 times. We observe that the data points shift to the right as the input noise magnitude increases, leading to more experiments with higher loss in output accuracy.

Sensitivity to Inputs: The sensitivity of accuracy loss to both the size and the data spread of inputs may vary considerably depending on the approximation technique and the application domain. Accordingly, different inputs should be considered in quantifying the complex trade-off space of accuracy vs. energy efficiency.

2.6. Metric Scope

We can deploy accuracy metrics at different scopes. For example, if multiple kernels constitute the application, an accuracy metric may apply for each kernel. This type of analysis is useful in characterizing the sensitivity of different phases of computation to noise, in order to facilitate selective approximation. However, derivation of the accuracy of the entire application from restricted-scope, per kernel metrics may not always be viable. This is because, depending on the particular execution trajectory, the impact of the degradation in the output of a kernel may grow, vanish, or stay the same as we reach the outputs of the encompassing application.

2.7. Acceptability vs. Accuracy

Accuracy metrics cannot measure "acceptability" of an execution outcome, which strongly depends on the context in which the corresponding application is deployed. Accordingly, context-oblivious analyses – as it is the case for most architectural studies – should deliver trade-off spaces, pareto fronts or ranges rather than randomly sampled points.

3. Metric Space

By exploiting semantic algorithmic information, we can refine generic accuracy metrics, or devise new ones. For a given application, many valid accuracy metrics may exist, possibly of different numeric stability.

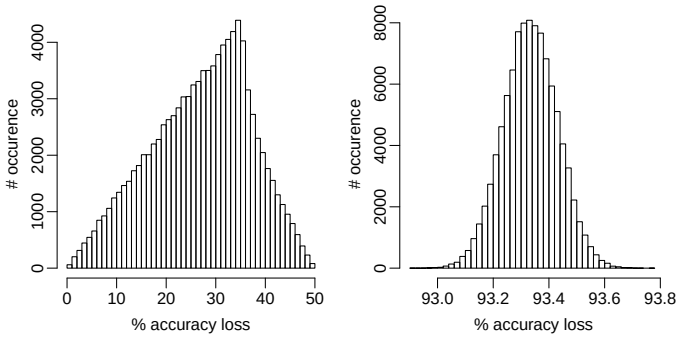
Case Study 11: In Figure 1a, we use a distance-based *distortion* metric to capture accuracy loss. Let us call this metric $M_{distance}$. The *deviation* corresponds to the absolute distance between *exact* and *approximate* positions of a particle normalized to the maximum possible distance (d_{max}) within the simulated space. Examining the x-axis, we observe that % accuracy loss captured by $M_{distance}$ remains less than 0.1% across all experiments. The worst-case degradation – 100% loss in accuracy – corresponds to all particle positions under *approximation* becoming distance d_{max} apart from their *exact* positions. This scenario is pretty unlikely in that it confines all *exact* and *approximate* positions to the lower and upper corners of the simulated space. On the other hand, $M_{distance}$ can capture *invalid* outputs. $M_{distance}$ assigns $> 100\%$ accuracy loss to particles falling outside the simulated space, since the position of such particles under *approximation* may shift further apart from their *exact* value than d_{max}. In order to mitigate the shortcomings of $M_{distance}$, we can devise an alternative metric, M_{cell}, to exploit algorithmic information in the following way: *fluidanimate* divides the simulated space into grids, where the granularity – the size of each grid cell – represents an application-specific knob. Each grid cell contains a fixed number of particles. Capturing interactions between the (centers of mass of) cells as opposed to individual particles notably reduces algorithmic complexity. We can record the grid cell encompassing each particle at the end of *exact* and *approximate* executions. M_{cell} then reports the number of mismatched particle-cell assignments between two executions, divided by the total number of particles in the simulated space. While semantically valid, M_{cell} misses how far particles moved away from their *exact* position under *approximation*: There is no distinction between the accuracy of two outcomes with one displaced particle only, if the displaced particle moved one cell-away in the first case, and by many cells-away, in the second. A new metric, $M_{cell-distance}$, can distinguish between similar cases: This time, we use the distance between the grid cells encompassing *exact* and *approximate* positions for each displaced particle, divided by the total number of cells. For the very same experiments conducted to generate the data in Figure 1a, Figures 1b and 1c capture the corresponding outcome under M_{cell}, and $M_{cell-distance}$, respectively. Overall, we observe that all of the three metrics render very similar differences in the accuracy loss of outputs across different experiments.

Case Study 12: Figure 4a captures the accuracy loss of *kmeans* under relaxed synchronization, deploying $M_{cluster-assign}$, the ratio of points which were assigned to a different cluster under *approximation*, when compared to the

exact outcome. For the very same execution, Figure 4b depicts the outcome under the alternative metric, $M_{cluster-center}$. $M_{cluster-center}$ instead captures the deviation of the cluster centers from their *exact* positions under *approximation*. This metric is very similar to the $M_{distance}$ for *fluidanimate*. Figures 5a and Figures 5b tabulate how these two metrics compare under precision reduction. Both of the metrics are semantically valid. The preference is likely to be determined by what kind of problem kmeans is solving. The metrics render notably different (similar) outcomes under approximate synchronization (precision reduction).

4. *Output Randomization* for Metric Evaluation

Accuracy loss under *approximation*, a bare number as calculated by an application-specific metric, oftentimes cannot give us a clear picture of how severe the degradation actually is. In other words, this number alone fails to provide enough insight into whether the observed accuracy loss is *acceptable*. To address this, we introduce an evaluation method for accuracy loss, *output randomization*. The idea is, for a given application-specific metric, (i) to generate a statistically significant number of completely *randomized* outputs; (ii) to calculate the accuracy loss for each completely *randomized* output, and (iii) to compare the accuracy loss under *approximation* with the average accuracy loss observed across *randomized* outputs. Being totally independent from the inputs, the average accuracy loss of *randomized* outputs is likely to come close to the worst-case.



(a) *particlefilter* (b) *kmeans*

Figure 8: % accuracy loss under *output randomization*.

Case Study 13: We consider two applications, *particlefilter* (from Rodinia) and *kmeans*. We generate 10^5 random outputs per se. *particlefilter* statistically derives the location of a target object from noisy measurements of the location. The output is the position of the object being tracked within the simulated, 128×128 frame. We generate random numbers for the x and y coordinates of the location within this simulated range. As the accuracy metric, we deploy the displacement of the point with respect to the *exact* position, divided by the diagonal length of the simulated frame. The *exact* position is at the center of the frame. Figure 8a depicts the distribution of the calculated % accuracy loss across 10^5 experiments. We observe that the % accuracy loss under *output randomization* always stays below 50%, as the points of maximum possible displacement under *randomization* (with respect to the *exact* position) can only be on the corners of the frame, excluding *invalid* outputs. The average is 27.1%. Figure 8b reflects the outcome under *output randomization* for *kmeans*. In this case, we generate random assignments for 65K points to 15 clusters. Contrary to *particlefilter*, the % accuracy loss under *output randomization* becomes 93.3% on average. Let us assume that we evaluate an approximation technique for both of these applications, and that the corresponding accuracy metrics render a % accuracy loss of 15% for both. Based on our results for *output randomization*, this 15% is more likely to correspond to an *acceptable* outcome for *kmeans* than for *particlefilter*.

Output randomization is an effective technique to reason about the severity of a given accuracy loss under *approximation*, as calculated by an application-specific metric. The idea is, for a given application-specific metric, to compare the accuracy loss under *approximation* with the average accuracy loss incurred by completely *randomized* outputs. Being totally independent from the inputs, *randomized* outputs are likely to incur a close-to-worst-case accuracy loss.

5. Putting It All Together

Based on our discussion in the previous sections, we next provide practical guidelines for quantitative characterization of accuracy loss under *approximation*, to cover metric selection, the design of experiments, and how to read the outcome of experiments.

5.1. On Metric Selection

In Table 2 we provide a compilation of (relative) metrics conforming to our guidelines from Sections 2 and 3. In accordance with Table 1, we adopt relative *deviation* (Section 1) in the output value for Class I. For Class II, on the other hand, we suggest two safe alternatives. The first one, *relative displacement*, reports the average displacement (across all elements) divided by the span of the simulated space (i.e. the diagonal length in a two dimensional simulated space). This metric is safe for applications that generate coordinate-based outputs. The second one, *Average Noise to Peak Signal*, ANPS, is defined as the average per element *deviation* normalized to the peak signal (i.e. maximum value) observed in the *exact* output. Both, ANPS and *distortion* in their basic definition assume that all elements of the output are of equal criticality. We can extend both metrics to incorporate per element weights to capture divergence in criticality – provided that such weights exist – by converting the average (arithmetic mean) to weighted average. In accordance with Table 1, we

Class	Output Data Type	Accuracy Metric	Domain	Examples
I	Numeric: scalar	Relative deviation in output value	Optimization	canneal (Parsec)
			Compression	dedup (Parsec)
II	Numeric: multi-dimensional	Relative displacement	n-body simulation	fluidanimate (Parsec) barnes, water (Splash2)
			Computer Vision	bodytrack (Parsec) particlefilter (Rodinia)
		Avg. Noise to Peak Signal (ANPS)	Linear Algebra	cholesky, lu (Splash2)
			Histogram	histo, tpacf (Parboil)
III	Compound (vector)	Relative mismatch	Clustering	streamcluster (Parsec)
				kmeans (Stamp)
			Data Mining	UtilityMine (MineBench)
			Similarity search	ferret (Parsec)
		Relative positional error	Sorting	radix (Splash2)
IV	Multi-media	SSIM	Computer Vision	raytrace (Splash2) volrend (Splash2)
			Video Encoding	x264 (Parsec)

Table 2: Compilation of (relative) accuracy metrics conforming to the guidelines from Sections 2 and 3.

adopt *relative mismatch*, and *positional error* for Class III. *Relative mismatch* is particularly useful if we do not care about the magnitude of each element, and all that matters is mismatches between elements under *approximate* and *exact* executions, respectively. *Positional error* applies to sorting applications, and quantifies the number of mis-ordered elements under *approximation*. Finally, in accordance with Table 1, we adopt SSIM for Class IV. The compilation in Table 2 provides only relative metrics. However, it may not be always possible, and moreover, necessary, to derive relative metrics. Absolute (i.e. non-normalized) versions of the metrics from Table 2 can as well serve quantification of the accuracy loss under *approximation*.

5.2. On the Design of Experiments

As approximation can easily induce non-determinism, we have to repeat experiments under the very same configuration for a statistically significant number of times. Only then we can extract the distribution of the accuracy loss under a given experimental setup (e.g. in histogram format), to be able to reliably compare different approximation scenarios. We should also capture the tails of such distributions (the extreme values of accuracy loss) in developing and evaluating techniques to bound the approximation-induced accuracy loss. In this context, trying to represent the entire distribution as a single numeric value (mean or median) is very likely to mask useful variation information.

The application domains which lend themselves well to approximation are of probabilistic nature, which incurs inherent non-determinism even if we exclude approximation. Accordingly, we should first characterize this inherent non-determinism to be able to distinguish approximation-induced degradation without any ambiguity. Different applications and approximation techniques render different spectra for the execution outcome: no program termination, termination with consistently *invalid* outputs, *valid* outputs spanning a wide accuracy loss interval, or some mixture thereof (Section 2.4). *Valid* outputs determine the trade-off space of accuracy vs. energy efficiency. The presence of non-terminating and *invalid* executions, on the other hand, necessitate safety nets for approximation to be viable. The energy overhead of such safety nets evolve with their complexity, and can easily alter the trade-off space. For example, if we devise re-execution as the safety net to recover from *invalid* or non-terminating executions, we should factor in the overhead of re-execution – weighed by the expected frequency of occurrence of *invalid* or non-terminating executions – in extracting the trade-off space.

5.3. How to Read the Outcome of Experiments?

Even if we deploy relative accuracy metrics (Section 2.2), we need to be very careful in comparing the accuracy loss under *approximation* across different applications. Different applications use diverse accuracy metrics, which can easily contaminate comparative analysis. Besides, as demonstrated in Section 4, the very same % accuracy loss may not translate into the exact same degree of degradation for different applications. In other words, while 15% can be highly *acceptable* for one application, it can be considered catastrophic for the other. The same story applies to comparison across different inputs. An application's noise tolerance can considerably change with different inputs, as shown in Section 2.5. Ergo, averaging across different inputs is not acceptable. Showing the efficiency of an approximation technique in bounding the accuracy loss *separately* for each application and input is more meaningful than targeting a single, fixed value of accuracy loss across different applications and inputs.

6. Conclusion

Approximate computing represents a promising paradigm in improving the energy efficiency, thus performance, of the

severely power-limited computing platforms of today. However, to be able to enable this potential, we need robust means to characterize the complex trade-off space of energy efficiency vs. accuracy. This article covers pitfalls and fallacies in the development and deployment of application-specific accuracy metrics which serve the purpose. Accuracy metrics from Table 2 along with the *Output Randomization* tool from Section 4 are publicly available for download at `altai.ece.umn.edu/accurax`.

References

[1] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," Princeton University, Tech. Rep. TR-811-08, January 2008.

[2] M. Breuer, "Hardware That Produces Bounded Rather Than Exact Results," in *Design Automation Conference (DAC)*, June 2010.

[3] S. T. Chakradhar and A. Raghunathan, "Best-effort Computing: Rethinking Parallel Software and Hardware," in *Design Automation Conference (DAC)*, 2010.

[4] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2009.

[5] V. Chippa, D. Mohapatra, and A. Raghunathan, "Scalable Effort Hardware Design: Exploiting Algorithmic Resilience for Energy Efficiency," *Design Automation Conference (DAC)*, June 2010.

[6] H. Cho, L. Leem, and S. Mitra, "ERSA: Error Resilient System Architecture for Probabilistic Applications," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 4, 2012.

[7] M. de Kruijf and K. Sankaralingam, "Exploring the Synergy of Emerging Workloads and Silicon Reliability Trends," in *SELSE*, 2009.

[8] M. de Kruijf, S. Nomura, and K. Sankaralingam, "Relax: An Architectural Framework for Software Recovery of Hardware Faults," in *International Symposium on Computer Architecture (ISCA)*, 2010.

[9] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture Support for Disciplined Approximate Programming," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012.

[10] U. Karpuzcu, I. Akturk, and N. S. Kim, "Accordion: Toward Soft Near-Threshold Voltage Computing," in *International Symposium on High Performance Computer Architecture (HPCA)*, February 2014.

[11] X. Li and D. Yeung, "Application-Level Correctness and its Impact on Fault Tolerance," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2007.

[12] J. S. Miguel, M. Badr, and N. E. Jerger, "Load Value Approximation," in *International Symposium on Microarchitecture (MICRO)*, December 2014.

[13] C. C. Minh, J. Chung, C. Kozyrakis, and K. Olukotun, "Stamp: Stanford transactional applications for multi-processing," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2008.

[14] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard, "Quality of Service Profiling," in *International Conference on Software Engineering (ICSE)*, 2010.

[15] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary, "MineBench: A Benchmark Suite for Data Mining Workloads," in *International Symposium on Workload Characterization*, October 2006.

[16] L. Renganarayana, V. Srinivasan, R. Nair, and D. Prener, "Programming with Relaxed Synchronization," in *ACM Workshop on Relaxing Synchronization for Multicore and Manycore Scalability (RACES)*, 2012.

[17] M. Rinard, "Parallel Synchronization-Free Approximate Data Structure Construction," *5th USENIX Workshop on Hot Topics in Parallelism*, 2013.

[18] M. C. Rinard, "Using Early Phase Termination to Eliminate Load Imbalances at Barrier Synchronization Points," in *Conference on Object-oriented Programming Systems and Applications (OOPSLA)*, 2007.

[19] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate Data Types for Safe and General Low-power Computation," in *Conference on Programming Language Design and Implementation (PLDI)*, 2011.

[20] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing Performance vs. Accuracy Trade-offs with Loop Perforation," in *European Software Engineering Conference and European Conference on Foundations of Software Engineering (ESEC/FSE)*, 2011.

[21] J. Sloan, D. Kesler, R. Kumar, and A. Rahimi, "A Numerical Optimization-based Methodology for Application Robustification: Transforming Applications for Error Tolerance," in *International Conference on Dependable Systems and Networks (DSN)*, 2010.

[22] J. A. Stratton, C. Rodrigrues, I.-J. Sung, N. Obeid, L. Chang, G. Liu, and W.-M. W. Hwu, "Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing," University of Illinois at Urbana-Champaign, Urbana, Tech. Rep. IMPACT-12-01, Mar. 2012.

[23] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, April 2004.

[24] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *International Symposium on Computer Architecture (ISCA)*, 1995.

[25] T. Y. Yeh, G. Reinman, S. J. Patel, and P. Faloutsos, "Fool Me Twice: Exploring and Exploiting Error Tolerance in Physics-based Animation," *ACM Transactions on Graphics*, vol. 29, December 2009.