Appendix - Aloe: Verifying Reliability of Approximate Programs in the Presence of Recovery Mechanisms

KEYUR JOSHI, University of Illinois at Urbana-Champaign, USA
VIMUTH FERNANDO, University of Illinois at Urbana-Champaign, USA
SASA MISAILOVIC, University of Illinois at Urbana-Champaign, USA

Modern hardware is becoming increasingly susceptible to silent data corruptions. As general methods for detection and recovery from errors are time and energy consuming, selective detection and recovery are promising alternatives for applications that have the freedom to produce results with a variable level of accuracy. Several programming languages have provided specialized constructs for expressing detection and recovery operations, but the existing static analyses of safety and quantitative analyses of programs do not have the proper support for such language constructs.

This work presents Aloe, a quantitative static analysis of reliability of programs with recovery blocks – a construct that checks for errors, and if necessary, applies the corresponding recovery strategy. The analysis supports reasoning about both reliable and potentially unreliable detection and recovery mechanisms. It implements a novel precondition generator for recovery blocks, built on top of Rely, a state-of-the-art quantitative reliability analysis for imperative programs. Aloe can reason about programs with scalar and array expressions, if-then-else conditionals, and bounded loops without early exits. The analyzed computation is idempotent and the recovery code re-executes the original computation.

We implemented Aloe and applied it to a set of eight programs previously used in approximate computing research. Our results present significantly higher reliability and scale better compared to the existing Rely analysis. Moreover, the end-to-end accuracy of the verified computations exhibits only small accuracy losses.

CCS Concepts: • **Theory of computation → Program specifications**; **Program verification**.

Additional Key Words and Phrases: Reliability, Approximate Computing

ACM Reference Format:
Keyur Joshi, Vimuth Fernando, and Sasa Misailovic. 2020. Appendix - Aloe: Verifying Reliability of Approximate Programs in the Presence of Recovery Mechanisms. In *Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization (CGO '20), February 22–26, 2020, San Diego, CA, USA*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3368826.3377924

CGO '20, February 22–26, 2020, San Diego, CA, USA
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-7047-9/20/02…$15.00
https://doi.org/10.1145/3368826.3377924

n	$\in$	$\mathbb{N}$	*quantities*
m	$\in$	$\mathbb{N} \cup \mathbb{F}$	*values*
r	$\in$	$[0, 1.0]$	*probability*
x, b	$\in$	Var	*variables*
a	$\in$	ArrVar	*array variables*
f	$\in$	Func	*external functions*
op	$\in$	$\{+, -, \dots\}$	*arithmetic operators*
Exp	$\rightarrow$	$m \mid x \mid f(Exp^*) \mid (Exp) \mid Exp\ op\ Exp$	*expressions*
t	$\rightarrow$	`int<n>`|`float<n>`	*basic types*
D	$\rightarrow$	t x | t a[n^+] | $D; D$	*variable declarations*
P	$\rightarrow$	$D; S$	*program*

recovery $\rightarrow$	
redo[n]	*redo up to n times*
| redo[ψ]	*redo on different reliability model*
| S	*other (custom) recovery*

$S \rightarrow$	
`skip`	*empty program*
| x = Exp	*assignment*
| x = Exp [r] Exp	*probabilistic choice*
| $S; S$	*sequence*
| x = a[Exp^+]	*array load*
| a[Exp^+] = Exp	*array store*
| if Exp {S} else {S}	*branching*
| repeat n {S}	*repeat n times*
| x = (T)Exp	*cast*
| try {S} check {Exp} recover {recovery}	*try-check-recover*

Fig. 1. Syntax

APPENDIX A

In this appendix we present the full semantics for our language. This language consists of the sequential subset from [3].

1 DEFINITIONS

Figure 1 shows the syntax of the language. In this section we define key terms and the key definitions.

References. A *reference* is a pair $\langle n_b, \langle n_1, \dots, n_k \rangle\rangle \in \texttt{Ref}$ that consists of a base address $n_b \in Loc$ and a dimension descriptor $\langle n_1, \dots, n_k \rangle$. References describe the location and the dimension of variables in the heap.

Frames, Stacks, and Heaps. A *frame* σ is an element of the domain $\mathrm{E} = \mathrm{Var} \rightarrow \mathrm{Ref}$ which is the set of finite maps from program variables to references. A *heap* $h \in H = \mathbb{N} \rightarrow \mathbb{N} \cup \mathbb{F}$ is a finite map from addresses (integers) to values. Values can be integers or floats. An environment $\epsilon \in \mathrm{E} \times H$ is a pair of a frame and a heap.

Programs. An approximated program executes within *approximation model*, ψ, which in general may contain the parameters for approximation (e.g., probability of selecting original or approximate expression). We define special reliable model 1_ψ, which evaluates the program without approximations.

Language Semantics. Figure 2 defines the semantics for expressions. Figures 3 and 4 define the semantics for statements.

Statements. The small-step relation $\boxed{\langle s, \sigma, h \rangle \xrightarrow{\lambda, p}_{\psi} \langle s', \sigma', h' \rangle}$ defines the program evaluating in a stack frame σ, and heap h with the transition label λ. The semantics of Aloe follow from Rely.

A *transition label* $\lambda \in \{\mathrm{C}, \mathrm{F}\}$ characterizes whether an error occurred (F) or not (C).

E-Var-C
$$\frac{\langle n_b, \langle 1 \rangle \rangle = \sigma(x)}{\langle x, \sigma, h \rangle \rightarrow_{\psi} \langle h(n_b), \sigma, h \rangle}$$

E-Var-F
$$\frac{\langle n_b, \langle 1 \rangle \rangle = \sigma(x)}{\langle x, \sigma, h \rangle \xrightarrow{1}_{\psi} \langle n_f, \sigma, h \rangle}$$

E-Iop-R1
$$\frac{\langle e_1, \sigma, h \rangle \xrightarrow{p}_{\psi} \langle e_1', \sigma, h \rangle}{\langle e_1 \; op \; e_2, \sigma, h \rangle \xrightarrow{p}_{\psi} \langle e_1' \; op \; e_2, \sigma, h \rangle}$$

E-Iop-R2
$$\frac{\langle e_2, \sigma, h \rangle \xrightarrow{p}_{\psi} \langle e_2', \sigma, h \rangle}{\langle n \; op \; e_2, \sigma, h \rangle \xrightarrow{p}_{\psi} \langle n \; op \; e_2', \sigma, h \rangle}$$

E-Iop-C
$$\frac{}{\langle n_1 \; op \; n_2, \sigma, h \rangle \xrightarrow{1}_{\psi} \langle op(n_1, n_2), \sigma, h \rangle}$$

Fig. 2. Dynamic Semantics of Expressions

Dec-Var
$$\frac{\langle n_b, h' \rangle = \mathsf{new}(h, \langle 1 \rangle)}{\langle \mathrm{T}\ \mathrm{x}, \sigma :: \sigma, h \rangle \xrightarrow{C,1}_{\psi} \langle \mathtt{skip}, \sigma[x \mapsto \langle n_b, \langle 1 \rangle \rangle] :: \sigma, h' \rangle}$$

Dec-Array
$$\frac{\forall i. 0 < n_i \qquad \langle n_b, h' \rangle = \mathsf{new}(h, \langle n_1 \ldots n_k \rangle) \qquad \sigma' = \sigma[x \mapsto \langle n_b, \langle n_1 .. n_k \rangle \rangle]}{\langle \mathrm{T}\ \mathrm{x}[n_1 \ldots n_k], \sigma, h \rangle \xrightarrow{C,1}_{\psi} \langle \mathtt{skip}, \sigma', h' \rangle}$$

Fig. 3. Semantics of Declarations

S-Assign-R
$$\frac{\langle e,\sigma,h\rangle \xrightarrow{p}_{\psi} \langle e',\sigma,h\rangle}{\langle x = e,\sigma,h\rangle \xrightarrow{C,p}_{\psi} \langle x = e',\sigma,h\rangle}$$

S-Assign-C
$$\frac{\langle n_b,\langle 1\rangle\rangle = \sigma(x)}{\langle x = n,\sigma,h\rangle \xrightarrow{C,1}_{\psi} \langle \mathtt{skip},\sigma,h[n_b \mapsto n]\rangle}$$

S-Assign-Prob-True
$$\frac{}{\langle x = e_1\ [r]\ e_2,\sigma,h\rangle \xrightarrow{C,r}_{\psi} \langle x = e_1,\sigma,h\rangle}$$

S-Assign-Prob-False
$$\frac{}{\langle x = e_1\ [r]\ e_2,\sigma,h\rangle \xrightarrow{F,1-r}_{\psi} \langle x = e_2,\sigma,h\rangle}$$

S-Assign-Approx-True
$$\frac{\langle l,\langle 1\rangle\rangle = \sigma(b) \qquad h[l] \neq 0}{\langle x = e_1\ [b]\ e_2,\sigma,h\rangle \xrightarrow{C,1}_{\psi} \langle x = e_1,\sigma,h\rangle}$$

S-Assign-Approx-True
$$\frac{\langle l,\langle 1\rangle\rangle = \sigma(b) \qquad h[l] = 0}{\langle x = e_1\ [b]\ e_2,\sigma,h\rangle \xrightarrow{C,1}_{\psi} \langle x = e_2,\sigma,h\rangle}$$

S-Seq-R1
$$\frac{\langle s_1,\sigma,h\rangle \xrightarrow{\lambda,p}_{\psi} \langle s_1',\sigma',h'\rangle}{\langle s_1\,;s_2,\sigma,h\rangle \xrightarrow{\lambda,p}_{\psi} \langle s_1'\,;s_2,\sigma',h'\rangle}$$

S-Seq-R2
$$\frac{}{\langle \mathtt{skip}\,;s_2,\sigma,h\rangle \xrightarrow{C,1}_{\psi} \langle s_2,\sigma,h\rangle}$$

S-If
$$\frac{\langle e,\sigma,h\rangle \xrightarrow{p}_{\psi} \langle e',\sigma,h\rangle}{\langle \mathtt{if}\ e\ \{s_1\}\ \mathtt{else}\ \{s_2\},\sigma,h\rangle \xrightarrow{C,1}_{\psi} \langle \mathtt{if}\ e'\ \{s_1\}\ \mathtt{else}\ \{s_2\},\sigma,h\rangle}$$

S-If-True
$$\frac{n \neq 0}{\langle \mathtt{if}\ n\ \{s_1\}\ \mathtt{else}\ \{s_2\},\sigma,h\rangle \xrightarrow{C,1}_{\psi} \langle s_1,\sigma,h\rangle}$$

S-If-False
$$\frac{n = 0}{\langle \mathtt{if}\ n\ \{s_1\}\ \mathtt{else}\ \{s_2\},\sigma,h\rangle \xrightarrow{C,1}_{\psi} \langle s_2,\sigma,h\rangle}$$

S-Array-Load-Idx
$$\frac{\langle e_i,\sigma,h\rangle \xrightarrow{p}_{\psi} \langle e_i',\sigma,h\rangle}{\langle x = a[n_1,\ldots,e_i,\ldots,e_k],\sigma,h\rangle \xrightarrow{C,p}_{\psi} \langle x = a[n_1,\ldots,e_i',\ldots,e_k],\sigma,h\rangle}$$

S-Array-Load-C
$$\frac{\langle n_b,\langle l_1,\ldots,l_k\rangle\rangle = \sigma(x) \qquad n_o = l_k + \Sigma_{i=0}^{k-1} n_i \cdot l_i \qquad n = h(n_b + n_o)}{\langle x = a[n_1,\ldots,n_k],\sigma,h\rangle \xrightarrow{C,p}_{\psi} \langle x = n,\sigma,h\rangle}$$

S-Array-Store-Idx
$$\frac{\langle e_i,\sigma,h\rangle \xrightarrow{p}_{\psi} \langle e_i',\sigma,h\rangle}{\langle a[n_1,\ldots,e_i,\ldots,e_k] = x,\sigma,h\rangle \xrightarrow{C,p}_{\psi} \langle a[n_1,\ldots,e_i',\ldots,e_k] = x,\sigma,h\rangle}$$

S-Array-Store-C
$$\frac{\begin{array}{c}\langle n_b,\langle l_1,\ldots,l_k\rangle\rangle = \sigma(x) \qquad n_o = l_k + \Sigma_{i=0}^{k-1} n_i \cdot l_i \\ \langle n_b',\langle 1\rangle\rangle = \sigma(x) \qquad h[n_b'] = v \qquad \psi(wr(m)) = 1\end{array}}{\langle a[n_1,\ldots,n_k] = x,\sigma,h\rangle \xrightarrow{C,1}_{\psi} \langle skip,\sigma,h[(n_b + n_o) \mapsto v]\rangle}$$

S-Try
$$\frac{\langle \mathrm{S1},\sigma,h\rangle \xrightarrow{\lambda,r}_{\psi} \langle \mathrm{S1}',\sigma',h'\rangle}{\begin{array}{l}\langle \text{try \{S1\} check \{e\} recover \{S2\}},\sigma,h\rangle \\ \qquad\qquad \xrightarrow{\lambda,r}_{\psi} \langle \text{try \{S1'\} check \{e\} recover \{S2\}},\sigma',h'\rangle\end{array}}$$

S-Check-1
$$\frac{\langle e,\sigma,h\rangle \xrightarrow{r}_{\psi} \langle e',\sigma,h\rangle}{\begin{array}{l}\langle \text{try \{}\mathtt{skip}\text{\} check \{e\} recover \{S2\}},\sigma,h\rangle \\ \qquad\qquad \xrightarrow{C,r}_{\psi} \langle \text{try \{}\mathtt{skip}\text{\} check \{e'\} recover \{S2\}},\sigma,h\rangle\end{array}}$$

S-Check-True
$$\frac{}{\langle \text{try \{}\mathtt{skip}\text{\} check \{true\} recover \{S2\}},\sigma,h\rangle \xrightarrow{C,1}_{\psi} \langle \mathtt{skip},\sigma,h\rangle}$$

S-Check-False
$$\frac{}{\langle \text{try \{}\mathtt{skip}\text{\} check \{false\} recover \{S2\}},\sigma,h\rangle \xrightarrow{C,1}_{\psi} \langle S2,\sigma,h\rangle}$$

Fig. 4. Semantics of Statements

APPENDIX B

1 SEMANTICS OF RELIABILITY

Aggregate semantics. We use the following aggregate semantics from Rely to define the reliability of a program.

Definition 1 (Trace Semantics for Programs).

$$\langle \cdot, \epsilon \rangle \overset{\tau,p}{\Longrightarrow}_{\psi} \epsilon' \equiv \langle \cdot, \epsilon.\sigma, \epsilon.h \rangle \overset{\lambda_1,p_1}{\longrightarrow}_{\psi} \ldots \overset{\lambda_n,p_n}{\longrightarrow}_{\psi} \langle \mathtt{skip}, \epsilon.\sigma, \epsilon.h \rangle$$
$$where\ \tau = \lambda_1,\ldots,\lambda_n,\ and\ p = \prod_{i=1}^{n} p_i$$

This big-step semantics is the reflexive transitive closure of the small-step global semantics for programs and records a *trace* of the program. The trace semantics are defined for environments (pairs of frames and heaps).

A trace $\tau \in T \rightarrow \cdot | \lambda :: T$ is a sequence of small step global transitions. The probability of the trace is the product of the probabilities of each transition.

Definition 2 (Aggregate Semantics for Programs).

$$\langle \cdot, \epsilon \rangle \Downarrow_{\psi}^{p} \epsilon'\ where\ p = \sum_{\tau \in T} p_{\tau}\ such\ that\ \langle \cdot, \epsilon \rangle \overset{\tau,p_{\tau}}{\Longrightarrow}_{\psi} \epsilon'$$

The big-step aggregate semantics enumerates over the set of all finite length traces and sums the aggregate probability that a program starts in an environment ϵ and terminates in an environment ϵ'. It accumulates the probability over all possible traces that end up in the same final state.

Paired Execution Semantics. For reliability and accuracy analysis we define a *paired execution semantics* that couples an original execution of a program with an approximate execution, following the definition from Rely.

Definition 3 (Paired Execution Semantics [2]).

$$\langle s, \langle \epsilon, \varphi \rangle \rangle \Downarrow_{\psi} \langle \epsilon', \varphi' \rangle\ such\ that\ \langle s, \epsilon \rangle \Downarrow_{1_{\psi}} \epsilon'\ and\ \varphi'(\epsilon_a') = \sum_{\epsilon_a \in E} \varphi(\epsilon_a) \cdot p_a\ where\ \langle s, \epsilon_a \rangle \Downarrow_{\psi}^{p_a} \epsilon_a'$$

This relation states that from a configuration $\langle \epsilon, \varphi \rangle$ consisting of an environment ϵ and an *environment distribution* $\varphi \in \Phi$, the paired execution yields a new configuration $\langle \epsilon', \varphi' \rangle$. The execution reaches the environment ϵ' from the environment ϵ with probability 1 (expressed by the deterministic execution, 1_{ψ}). The environment distributions φ and φ' are probability mass functions that map an environment to the probability that the execution is in that environment. In particular, φ is a distribution on environments before the execution of s whereas φ' is the distribution on environments after executing s.

Reliability Transformer. Reliability predicates and the semantics of programs are connected through the view of a program as a reliability transformer.

Definition 4 (Reliability Transformer Relation [2]).

$$\boxed{\psi \models \{Q_{pre}\}\, s\, \{Q_{post}\}} \equiv \forall \epsilon, \varphi, \epsilon', \varphi'.\ (\epsilon, \varphi) \in [\![Q_{pre}]\!] \implies \langle s, \langle \epsilon, \varphi \rangle \rangle \Downarrow_{\psi} \langle \epsilon', \varphi' \rangle \implies (\epsilon', \varphi') \in [\![Q_{post}]\!]$$

Similar to the standard Hoare triple relation, if an environment and distribution pair $\langle \epsilon, \varphi \rangle$ satisfy a reliability predicate Q_{pre}, then the program's paired execution transforms them into a new pair $\langle \epsilon', \varphi' \rangle$ that satisfy a predicate Q_{post}.

2 CORRECTNESS OF RELIABILITY PRECONDITION GENERATION FOR THE TRY-CHECK-RECOVER BLOCK

Theorem 1 (Correctness of Reliability Precondition Generation for the try-check-recover Block). *If $\psi \models \{c \leq r r_t \mathcal{R}(Y_t)\}\, s_{try}\, \{c \leq r\mathcal{R}(X)\}$, $\psi \models \{c \leq r r_r \mathcal{R}(Y_r)\}\, s_{rec}\, \{c \leq r\mathcal{R}(X)\}$, p_t is the minimum success probability of s_{try}, p_{TN}, p_{FP}, and p_{TP} are the relevant properties of the checker f, and s_{try} and s_{rec} satisfy the dataflow constraints and perform the same computation,*
then $\psi \models \{c \leq r(p_t p_{TN} + p_t p_{FP} r_r + (1-p_t) p_{TP} r_r)\mathcal{R}(Y_t \cup Y_r)\}$ try$\{s_{try}\}$check$\{f\}$recover$\{s_{rec}\}$ $\{c \leq r\mathcal{R}(X)\}$

Proof. To prove Theorem 1, we first replace the precondition of s_{try} with $c \leq r p_t \mathcal{R}(Y_t \cup Y_r)$ and that of s_{rec} with $c \leq r r_r \mathcal{R}(Y_t \cup Y_r)$. As $p_t \leq r_t$ and $Y_t, Y_r \subseteq Y_t \cup Y_r$, this is a sound replacement (Proposition 2 of [2]).

The variables in X and $Y_t \cup Y_r$ fall into one of three categories:

(1) Variables that are neither read nor written to by the try or recover blocks.
(2) Variables that are read by the try or recover blocks.
(3) Variables that are written to by the try or recover blocks.

The variables in category 1 are transferred from the postcondition's joint reliability predicate to the precondition's joint reliability predicate unchanged, as per Rely's precondition generation rules. Aloe requires that the try and recover blocks be idempotent – future executions of the try or recover blocks should not be affected by the current execution. Therefore, category 2 and 3 must be mutually exclusive – otherwise, variables written to in the current execution would affect future executions.

Let ϵ, φ be the environment and environment distribution before executing the try-check-recover block, and ϵ', φ' after executing the try-check-recover block. By definition, $[\![\mathcal{R}(X)]\!](\epsilon', \varphi') = \sum_{\epsilon_u \in \mathcal{E}(\{X\}, \epsilon')} \varphi'(\epsilon_u)$. We consider two ways in which we can reach an environment in which variables in X are calculated correctly ($\mathcal{E}(\{X\}, \epsilon')$):

(1) We start from an initial environment in which variables in Y_t have been calculated correctly, execute the try block, which calculates the variables in X correctly, and then the check passes.
(2) We start from an initial environment in which variables in Y_r have been calculated correctly, execute the try block, fail the check, and then execute the recover block, which calculates the variables in X correctly.

The total probability of reaching a state in $\mathcal{E}(\{X\}, \epsilon')$ is the sum of the probabilities of these two cases. For simplification, we can soundly replace Y_t, Y_r in the two cases with $Y_t \cup Y_r$. Then we assume we start from an environment in $\mathcal{E}(\{Y_t \cup Y_r\}, \epsilon)$. By definition, $[\![\mathcal{R}(Y_t \cup Y_r)]\!](\epsilon, \varphi) = \sum_{\epsilon_u \in \mathcal{E}(\{Y_t \cup Y_r\}, \epsilon)} \varphi(\epsilon_u)$.

Case 1: From the precondition / postcondition of the try block *in isolation*, we know that $\sum_{\epsilon_u \in \mathcal{E}(\{X\}, \epsilon')} \varphi'(\epsilon_u) \geq \sum_{\epsilon_u \in \mathcal{E}(\{Y_t \cup Y_r\}, \epsilon)} \varphi(\epsilon_u) \times p_t$. However, within the try-check-recover block, after a correct execution of the try block, the check passes with probability p_{TN}. Therefore the contribution of this case is $\sum_{\epsilon_u \in \mathcal{E}(\{Y_t \cup Y_r\}, \epsilon)} \varphi(\epsilon_u) \times p_t \times p_{TN}$.

Case 2: From the precondition / postcondition of the recover block *in isolation*, we know that $\sum_{\epsilon_u \in \mathcal{E}(\{X\}, \epsilon')} \varphi'(\epsilon_u) \geq \sum_{\epsilon_u \in \mathcal{E}(\{Y_t \cup Y_r\}, \epsilon)} \varphi(\epsilon_u) \times r_r$.

However, within the try-check-recover block, for this case, the check must fail. This happens in two ways: either the try block executes correctly and the check fails, or the try block causes an error and the check fails. The first sub-case happens with probability $p_t p_{FP}$ and the second sub-case with probability $(1-p_t)p_{TP}$. Therefore the contribution of this case is $\sum_{\epsilon_u \in \mathcal{E}(\{Y_t \cup Y_r\},\epsilon)} \varphi(\epsilon_u) \times r_r \times (p_t p_{FP} + (1-p_t)p_{TP})$. The idempotency constraint on the try block ensures that r_r is unchanged by the try block's probability of causing an error.

Combining the two cases: Adding up the probabilities of the two cases of the try-check-recover block execution, we finally get

$$\sum_{\epsilon_u \in \mathcal{E}(\{X\},\epsilon')} \varphi'(\epsilon_u) \geq \sum_{\epsilon_u \in \mathcal{E}(\{Y_t \cup Y_r\},\epsilon)} \varphi(\epsilon_u) \times p_t \times p_{TN} + \sum_{\epsilon_u \in \mathcal{E}(\{Y_t \cup Y_r\},\epsilon)} \varphi(\epsilon_u) \times r_r \times (p_t p_{FP} + (1-p_t)p_{TP})$$
$$\geq \sum_{\epsilon_u \in \mathcal{E}(\{Y_t \cup Y_r\},\epsilon)} \varphi(\epsilon_u)(p_t \times p_{TN} + r_r \times (p_t p_{FP} + (1-p_t)p_{TP}))$$

That is, $[\![\mathcal{R}(X)]\!](\epsilon',\varphi') \geq [\![\mathcal{R}(Y_t \cup Y_r)]\!](\epsilon,\varphi)(p_t \times p_{TN} + r_r \times (p_t p_{FP} + (1-p_t)p_{TP}))$.

□

APPENDIX C

1 EVALUATION

Table 1. Experimental Setup for Evaluation

Benchmark	Input
PageRank	10 Iterations, randomly generated graph with 1000 nodes
Scale	512 × 512 pixel image (baboon.ppm)
Blackscholes	4K option prices from the Parsec Inputs [1]
SSSP	randomly generated graph with 1000 nodes
BFS	randomly generated graph with 1000 nodes
SOR	10 iteration on a 1000×1000 array
Sobel	1000×1000 array in the range [0,1]
Motion	10 blocks with 1600 pixels each

REFERENCES

[1] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. PACT, 2008.

[2] Michael Carbin, Sasa Misailovic, and Martin C. Rinard. Verifying quantitative reliability for programs that execute on unreliable hardware. In *OOPSLA*, 2013.

[3] Vimuth Fernando, Keyur Joshi, and Sasa Misailovic. Verifying safety and accuracy of approximate parallel programs via canonical sequentialization. In *OOPSLA*, 2019.