Interval-Adjoint Significance Analysis: A Case Study

Jens Deussen, Jan Riehme and Uwe Naumann
LuFG Informatik 12, RWTH Aachen University
Software and Tools for Computational Engineering
52074 Aachen, Germany
Email: [deussen,riehme,naumann]@stce.rwth-aachen.de

***Abstract*—We report further development of the interval-adjoint significance analysis (IASA) as a part of the SCoRPiO project. In SCoRPiO significance based approximate computing is used to reduce the energy consumption of a program execution by tolerating less accurate results.**

Part of the project is to define significance as an algorithmic property to quantify the impact of a computation to the output. Information needed by this definition is obtained by an analysis combining algorithmic differentiation and interval arithmetic. Thus, the analysis can identify computations that can be evaluated less accurate, e.g. on low power but less reliable hardware.

An interval splitting approach is presented to address issues introduced by a naive usage of interval arithmetic. This approach additionally offers a more detailed and refined analysis of the underlying computer code. Furthermore, we introduce the quantification mode that is used to verify intuitive and well known characteristics of a code and to obtain the corresponding insignificant computations.

I. Introduction

The basic idea of approximate computing is to relax reliability constraints for the benefit of an energy efficient execution of a software. Approximate result are sufficient for a lot of applications, e.g. in the field of multimedia, audio, image and video processing respectively, or in data mining. SCoRPiO wants to exploit the fact that the computations of an applications has different importance for the quality of the result. Automatic code characterization techniques which use compile- and run-time analyses can be used to classify those computations as significant or insignificant. Significant computations are more important to the quality of the output while insignificant computations just have a minor impact. Tolerating a controlled degree of imprecision enables that insignificant computations can be steered to low power yet less reliable hardware. Furthermore, the significance information can be used on a software level to save computing time by replacing insignificant computations by representative expressions.

Other approaches to significance-driven approximate computing exist. In [1] statistical analysis is used to classify the computations into significant and non-significant. [2] obtains significance information by interval arithmetic and error propagation based on local partial derivatives.

As a part of the FET-open project SCoRPiO [1] we developed IASA to obtain significance information of the computations of a computer program for an user defined input domain. Therefore, the given `C` or `C++` source code is transformed to a directed acyclic graph representing single assignment code. Graph analysis can extend the significance information by identifying those computations that only lead to insignificant parts of the code. Finally, energy efficient code variants of the software are generated by using the significance information. Previous contributions to WAPCO and other conferences addressing IASA of SCoRPiO are [3], [4], [5].

The main idea of IASA, presented in [3], is to define a significance based on the forward propagation of interval values and the backward propagation of adjoints. Therefore, Interval Arithmetic [6] and Algorithmic Differentiation [7], [8], [9] are combined. Additionally, we define interval splitting [4] to address difficulties introduced by the naive usage of interval arithmetic, e.g. unfeasible relational operators or the *wrapping effect* [10]. This approach produces multiple scenarios and enables to generate even more efficient code variants for subdomains of the user defined input domain.

In this paper we will introduce two different types of interval splitting strategies, the exploration and the quantification mode respectively. While the exploration mode is a blackbox strategy analyzing code without additional information, the quantification mode verifies knowledge about the code and identifies insignificant parts of the code. As a case study we analyze a simulation of the 1-D heat equation with IASA. The quantification mode is used to verify that the influence of a heat source to points beyond a specific range is negligible for small simulation times.

The document is organized as follows: Section II outlines the methodology of IASA. We give a algorithmic definition of significance and motivate the interval splitting. Subsequently, the exploitation of significance information is described. In section III we analyze a simulation of the heat equation by IASA. Finally, section IV summarizes the document and gives an outlook.

II. Methodology

In this section we give an overview of the interval adjoint based significance and the mathematical concepts used in IASA. Furthermore, we present the tool implementing IASA and introduce an interval splitting approach addressing issues

[1] http://www.scorpio-project.eu/outline/

that occur in practice. For a more detailed description of IASA see e.g. [11].

For IASA we assume a computer program P implementing a multivariate scalar function $f : D \to I$ with domain $D \subseteq \mathbb{R}^n$, image $I \subseteq \mathbb{R}$ and $y = f(\mathbf{x})$ in which $\mathbf{x} = (x_1, \ldots, x_n)^T$. The implementation of function f can be decomposed into a sequence of p elemental functions (binary operations and intrinsic functions) by the three-part evaluation procedure from [8]. This transformation yields a single assignment code (SAC) that can be represented as a directed acyclic graph (DAG) $G = (V, E)$ with the set of nodes $V = 1, \ldots, n+p$ and directed edges $E = \{(j, i) | j \prec i\}$. The nodes represent the input, intermediate and output variables with their corresponding elemental function and each edge denotes a direct dependency $j \prec i$ of the i-th assignment on the result of the j-th computation

A significance analysis should assign significance values $S_y(v_i)$ for all intermediate variables v_i representing node i and for a given input domain $D = [\mathbf{x}] = [\underline{\mathbf{x}}, \overline{\mathbf{x}}] = \{\mathbf{x} \in \mathbb{R}^n | \underline{\mathbf{x}} \leq \mathbf{x} \leq \overline{\mathbf{x}}\}$ of the inputs with lower bound $\underline{\mathbf{x}} \in \mathbb{R}^n$ and upper bound $\overline{\mathbf{x}} \in \mathbb{R}^n$. Therefore, the influence of the input domain $[\mathbf{x}]$ on each intermediate variable v_i and the influence of each intermediate variable on the output y need to be quantified. The significance value should be a combination of both information.

A. Interval Adjoint Significance Analysis

Interval Arithmetic (IA) evaluates functions by using specified ranges instead of actual values. The interval evaluation $f[\mathbf{x}]$ of f for the given input range $[\mathbf{x}]$ yields a guaranteed enclosure $f[\mathbf{x}] \supseteq \{f(\mathbf{x}) | \mathbf{x} \in [\mathbf{x}]\}$ that contains all possible function values of $f(\mathbf{x})$ for $\mathbf{x} \in [\mathbf{x}]$. Additionally, enclosures $[v_i]$ for all intermediate variables v_i are obtained by the interval evaluation.

Nevertheless, a function evaluation in IA only yields a combination of the influence of all input and intermediate predecessors on an intermediate or output variable. To obtain the individual influence of intermediate variables for the final output y adjoint Algorithmic Differentiation can be used.

Algorithmic Differentiation (AD) computes additionally to the function value its derivatives by using the chain rule. The adjoint mode of AD can compute the first order derivatives $\frac{\partial y}{\partial v_i}$ of the output y with respect to all intermediate variables v_i by a single evaluation of an adjoint model of f.

A combination of IA and AD yields interval valued partial derivative $\nabla_{[v_i]}[y]$ that contains all possible derivatives of output y with respect to intermediate variable v_i for the given input domain $[\mathbf{x}]$.

B. Definition of Significance

A new definition for the significance of an intermediate variable v_i that also combines forward information of IA and backward information of AD is given for a defined input range $[\mathbf{x}]$ by

$$S_y(v_i) = w[v_i] \cdot \max |\nabla_{[v_i]}[y]|. \tag{1}$$

In (1) $w([v_i]) = \overline{v_i} - \underline{v_i}$ denotes the width of the interval value $[v_i]$ that measures the impact of the input $\mathbf{x}$ on an intermediates variable v_i. A large width $w([v_i])$ indicates that the intermediate v_i is highly sensitive to the variation of all inputs $\mathbf{x}$ in the range $[\mathbf{x}]$. Furthermore, the absolute maximum of the first order derivatives $|\nabla_{[v_i]}[y]|$ of output y with respect to intermediate v_i is a measurement for the individual influence of intermediate variables to the output. If the absolute value of this derivative $|\nabla_{[v_i]}[y]|$ is small, a change in the value of v_i has just a small impact on y.

Variables v_i with a significance value less or equal to a predefined significance bound ϵ are called insignificant:

$$S_y(v_i) \leq \epsilon$$

All other variables are called significant and have to be computed precisely.

The significance information extends the previously defined DAG to a significance DAG $G^S = (V^S \cup V^C, E)$ in which $V^S = \{i | S_y(v_i) > \epsilon, i \in V\}$ is a subset of V containing all significant nodes and V^C is the set of insignificant nodes with $V = V^S \cup V^C$. Applying IASA to a DAG G for a given input domain D will yield the corresponding significance DAG G^S.

The wrapping effect might yield overly pessimistic results by large overestimation of the value intervals $[v_i]$. An artificial significance is created by this overestimation due to the direct dependency of the significance on the width of those value intervals $w([v_i])$ in (1). Thus, the significance information quality gained by IASA is low in that case. Overestimation grows also with the size of the input domains. Moreover, in IA a comparison $[v_i] < c$ of an interval $[v_i]$ with a constant c is not defined if the interval contains the constant ($c \in [v_i]$). In the following section we introduce an interval splitting approach to address those issues.

C. Interval Splitting

The general idea of interval splitting is to divide value intervals $[v_i]$ into a predefined number of subinterval and generate multiple IASA scenarios. In case of a comparison $[v_i] < c$ with $c \in [v_i]$ the interval is split into two scenarios with $[\underline{v_i}, c)$ or $[c, \overline{v_i}]$. Thus, the interval splitting approach can be used to apply IASA to computer programs with arbitrary control flow.

We construct two different modes for the interval splitting, the exploration mode and the quantification mode respectively. The exploration mode is a sort of blackbox analysis, in which all variables are split that have a large value or adjoint interval. Thus, the user has to give an upper bound for the maximal allowed width of all value and adjoint intervals. In contrast, the quantification mode considers user knowledge about the code to find promising input domains in terms of computational savings. Only user selected input variables are split to obtain the significance values and thereby the insignificant parts of the code.

Interval splitting of the input domain yields the application of IASA to a subset $D_k \subseteq D$ of the original domain. A scenario $s_k = (G_k, D_k)$ is uniquely defined by its input

domain D_k and the corresponding DAG G_k. IASA is applied recursively to new scenarios.

D. Interpretation of Significant Information

In [3] we already proposed different possibilities to exploit insignificant parts of the code. For this case study we assume that all values of an interval have the same expectation such that insignificant variables can be replaced by constants equal to the center of the interval. Setting a variable v_i to a constant yields elimination of the edges connecting v_i with its predecessors in the significance DAG. Dependency analysis can be used to propagate insignificance information to predecessors and successors to replace or remove further nodes, e.g. *activity analysis* []. An advanced significance DAG $G^{S+} = (V^{S+} \cup V^{C+}, E)$ additionally stores this information with V^{C+} containing the indices of those nodes that can be saved by replacing them with constants and V^{S+} is the set of nodes that have to be computed precisely.

A significance DAG $G_k^S = (V_k^S \cup V_k^C, E_k)$ can be transformed back to source code by a simple extension of the interpreter for graph G^S, where nodes $v_i \in V^C$ are replaced by constants. Codes for advanced significance DAGs G_k^{S+} can be generated analogous with V_k^{S+} and V_k^{C+} instead of V_k^S and V_k^C, respectively. In case of interval splitting, we need to rank scenarios and we will only generate special code variants for the top-ranked scenarios to keep the size of the entire source code acceptable. Thus, we define the rank r_k of a scenario s_k as the product

$$r_k = r(G_k^S, D_k) = \prod_{i=1}^{n} w(D_k^i) \cdot |V_k^C|, \tag{2}$$

in which $|V_k^C|$ is the total number of insignificant variables and $\prod_{i=1}^{n} w(D_k^i)$ is the input domain size. This combines the probability to execute the scenario and the savings that are obtained by its execution.

E. Implementation

To make IASA available the user has to include the header file of IASA. All variables with a floating point data type have to be replaced by the special data type `siggraph::flat` of IASA. The user has to specify typical input ranges for a set of input variables and register them. Furthermore, the user has to define a set of output variables for the seeding of the adjoint intervals. For a given significance bound and the required parameters for the interval splitting IASA identifies the insignificant parts of the code and generates special code variants.

The implementation of IASA is based on a combination of the library dco/c++ [12] with an interval base type from the interval library filib++ [13] both written in `C++`.

III. Case Study

The well known 1-D heat equation describes the distribution of heat in a thin rod heated by a candle at the right end for time $t \leq t_f$. We use a rod length $L = 2$, a constant heat coefficient $c = 0.01$, and examine the temperature T at the center of the rod. Because the heat from the source will not reach the center of the rod for small simulation times t_f, there is a limited change in the temperature T only in the left part of the rod. Nevertheless, a numerical code would compute new (but nearly unchanged) values of temperature T for the left part of the rod. We use the quantification mode of IASA to verify that a lot of computations are insignificant in that case.

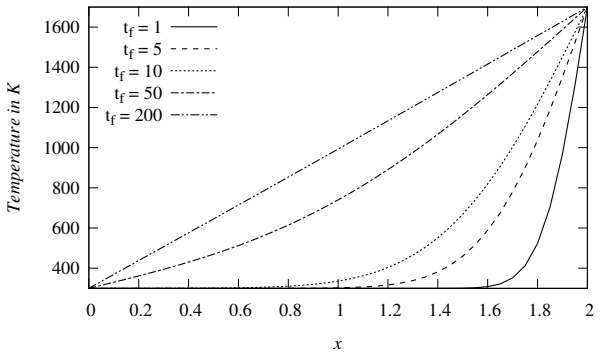

Fig. 1: Heat distribution for different simulation times

A. Problem Description

The distribution of heat within the rod over time is modeled by the parabolic partial differential equation

$$\frac{\partial T}{\partial t} = c \cdot \frac{\partial^2 T}{\partial x^2}, \qquad T = T(t, x, c), \tag{3}$$

where $x \in [0, 2]$ is a space variable and $t \in [0, t_f]$ denotes the time. The initial condition is $T(0, x, c) = 300K \ \forall x \in (0, 2)$ and the boundary conditions are $T(t, 0, c) = 300K$ and $T(t, 2, c) = 1700K \ \forall t \in [0, t_f]$. Solving the 1-D heat equation (3) for the given initial and boundary conditions yields a temperature distribution $T = T(t, x, c)$. The heat distribution for five different simulation times is visualized for the described test case in Fig. 1.

An equidistant spatial discretization of $x = (x_0, \ldots, x_{n-1})^T$ with $\Delta x = \frac{L}{n-1}$ in (3) with a second order finite difference scheme gives the ordinary differential equation (ODE)

$$\frac{\partial T_j}{\partial t} = \frac{c}{(\Delta x)^2} \cdot r_j(T) \ , \quad j = 0, \ldots, n-1 \ , \tag{4}$$

with

$$r_j = 0 \ , \qquad j \in \{0, n-1\} \tag{5}$$

$$r_j = T_{j+1} - 2 \cdot T_j + T_{j-1} \ , \qquad j \in \{1, \ldots, n-2\}, \tag{6}$$

where $T_j = T(t, x_j, c)$ is the temperature at position $x = x_j$.

Because (4) is linear in T a first order Taylor series at point $T \equiv 0$ ($T_j = 0$ for $j = 1, \ldots, n-2$) expands the ODE

$$\frac{\partial T}{\partial t} = \frac{c}{(\Delta x)^2} \cdot \left(\underbrace{r(0)}_{=0} + \frac{\partial r}{\partial T} \cdot (T - 0) \right). \tag{7}$$

The residual r at $T \equiv 0$ vanishes identically as a result of (5) and (6). Linearity of the residual in T implies the

independence of its first derivative from T, such that the Jacobian $\left(\frac{\partial r}{\partial T}\right)_{i,j}$ has the form

$$\left(\frac{\partial r}{\partial T}\right)_{i,j} = \begin{cases} -2, & \text{if } i = j \quad \wedge i \notin \{0, n-1\} \\ 1, & \text{if } |i-j| = 1 \wedge i \notin \{0, n-1\} \\ 0, & \text{else} \end{cases} \tag{8}$$

with row index i and column index j.

The left hand side of the ODE in (7) is approximated by a backward finite difference scheme yielding the backward Euler method. Let $T^k = T(k\Delta t, x, c)$ be the temperature at time $t = k\Delta t$ and for a temporal discretization with m time steps of equal length $\Delta t = \frac{t_f}{m}$, (7) transforms to

$$\frac{T^{k+1} - T^k}{\Delta t} = \frac{c}{(\Delta x)^2} \cdot \frac{\partial r}{\partial T} \cdot T^{k+1}, \tag{9}$$

which imply the linear system

$$\underbrace{I - a \cdot \frac{\partial r}{\partial T}}_{A} \cdot T^{k+1} = T^k, \tag{10}$$

with

$$A = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ -a & 1+2a & -a & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -a & 1+2a & -a \\ 0 & \cdots & 0 & 0 & 1 \end{pmatrix}, \tag{11}$$

and

$$a = \frac{c\Delta t}{\Delta x^2}. \tag{12}$$

In (10) I denotes the identity in $\mathbb{R}^n$ and A is the system matrix that is constant for given c, Δx and Δt.

B. Setup of the Significance Analysis

For the simulation we use a discretization with $m = 800$ points in time and $n = 41$ points in space. Furthermore, we choose the initial temperature of the rod $T(0, x, c)$, the temperature at the candle $T(t, 2, c)$ and the final time t_f as inputs. The input domains for those inputs are $T(0, x, c) \in [280, 300]$, $T(t, 2, c) \in [1650, 1700]$ and $t_f \in [0, 32]$. The output of the simulation is the temperature $T(t_f, 1, c)$ at the center of the rod at the final time $t = t_f$.

Listing 1 gives the implementation of the 1-D heat problem that is analyzed. In lines 3-12 the system matrix A is constructed once as shown in (11). Then, A is decomposed into a lower and an upper triangular matrix that are stored in the same variable as A in lines 14-21. The iteration over the time steps is implemented in line 23-36 in which the linear system from (10) is solved by a forward substitution in lines 24-28 and a backward substitution in lines 30-35. The inputs are registered in lines 48-50 right before the start of the simulation and the registration of the output is done in line 59. Finally, IASA is started in line 60.

Listing 1: Implementation of the simulation of the 1-D heat equation

```
1  template<typename Type>
2  void heat(int m, int n, Type& c, Type& x, Type& t,
       vector<Type>& T) {
3    siggraph::set_section("System Matrix");
4    vector<Type> A(n*n);
5    A[0] = 1;
6    A[n*n-1] = 1;
7    Type a = t*pow(n-1,2.0)*c/m/pow(x,2.0);
8    for (int i=1; i<n-1; i++) {
9      A[i*n+i-1] = -a;
10     A[i*n+i] = 2*a+1;
11     A[i*n+i+1] = -a;
12   }
13
14   siggraph::set_section("LU Decomposition");
15   for (int k=0; k<n; k++) {
16     for (int i=k+1; i<n; i++)
17       A[i*n+k] /= A[k*n+k];
18     for (int j=k+1; j<n; j++)
19       for (int i=k+1; i<n; i++)
20         A[i*n+j] -= A[i*n+k]*A[k*n+j];
21   }
22
23   for (int j=0; j<m; j++) {
24     siggraph::set_section("Forward Substitution");
25     for (int i=0; i<n; i++) {
26       for (int j=0; j<i; j++)
27         T[i] -= A[i*n+j]*T[j];
28     }
29
30     siggraph::set_section("Backward Substitution");
31     for (int i=n-1; i>=0; i--) {
32       for (int j=n-1; j>i; j--)
33         T[i] -= A[i*n+j]*T[j];
34       T[i] /= A[i*n+i];
35     }
36   }
37
38 }
39
40 int main(int argc, char* argv[]){
41   int n = atoi(argv[1]), m = atoi(argv[2]);
42   double pc = 0.01, L = 2, pT0 = 300, pTN = 1700;
43   double t_s = atof(argv[3]),t_f = atof(argv[4]);
44
45   siggraph::set_section("Initialization");
46   siggraph::flat c = pc, x = L, t = t_f;
47   siggraph::flat T0 = pT0, TN = pTN;
48   siggraph::register_input(t, t_s, t_f);
49   siggraph::register_input(TN, 1650, 1700);
50   siggraph::register_input(T0, 280, 300);
51
52   vector<siggraph::flat> T(n);
53   for (int i=0; i<n-1; i++) T[i] = T0;
54   T[n-1] = TN;
55
56   heat(m,n,c,x,t,T);
57
58   siggraph::set_section("Analysis");
59   siggraph::register_output(T[(n-1)/2], 1.0);
60   siggraph::analyse(1.0);
61   return 0;
62 }
```

C. Results

The code flattener generates a DAG with 2,702,061 nodes for the given implementation and setup in section III-B. With a significance bound of $\epsilon = 0$ IASA identifies 1,292,973 nodes to be insignificant which is about 48% of the total computations. Such a high ratio of insignificant computations for a relatively unspecific input domain indicates an inefficient code.

In this particular case a dense solver is used to compute the solution of the linear system with the tridiagonal system

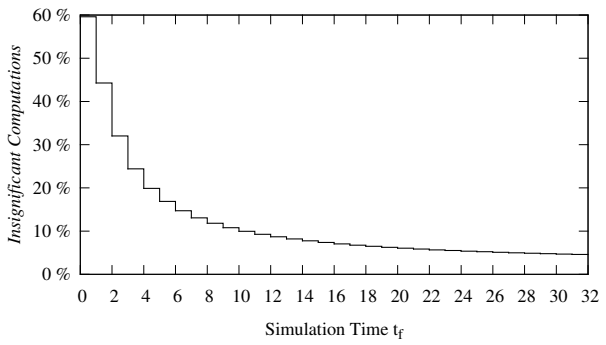

Fig. 2: Ratio of insignificant computations plotted over the simulation time t_f for an input domain with width equal to 1

matrix A. The dense LU decomposition has a complexity of $\mathcal{O}(n^3)$ while the dense forward and backward substitution have a complexity of $\mathcal{O}(n^2)$. The substitutions are called m-times which yields a total complexity of $\mathcal{O}(mn^2 + n^3)$. By exploiting the sparsity pattern of the system matrix and using a sparse LU decomposition and specialized forward and backward substitutions for tridiagonal matrices the computational complexity can be reduced to $\mathcal{O}(mn)$.

Replacing the dense linear solver in our implementation in Listing 1 by a sparse solver and LU decomposition the generated DAG only contains 161,081 nodes. Again using a significance bound of $\epsilon = 0$ yields 1,665 computations that are insignificant. These computations are related to the zeros in the first and the last row of system matrix A in (11) that are not exploited by using a sparse solver for tridiagonal matrices. Thus, IASA automatically detects the inefficiency of the implementation. We are aware that it is obvious to exploit the structure of the system matrix and use a sparse solver in this particular case, but nevertheless IASA automatically detects this lack of attention or knowledge.

In the following we use the quantification mode with a binary interval splitting only on the variable for the final time t_f. Binary interval splitting divides the interval into two equal sized subintervals. Furthermore, we assume a termination condition for the interval splitting: If the width of the interval that should be split is already lower or equal to 1 no further scenarios are generated.

The results show that analyzing input intervals with a smaller width yields a larger number of insignificant computations. This is induced by holding the significance bound ϵ constant because the significance is dependent on the width of the intervals: Using the significance definition from (1) the significance value drops by decreasing the width of the input domain. Furthermore, we can validate the expected result that a lot of computations are insignificant for small final simulation times t_f. This dependency is visualized in Fig. 2.

To get an idea which computations are marked as insignificant Fig. 3 shows the automatic generated advanced significance DAG for a problem size of $m = 3$ time steps and $n = 9$ distributions in space. The different node colors

Tab. 1: Top ten ranked scenarios of the simulation of the heat equation with their interval ranges for the final time t_f and the number of insignificant computations $|V_k^C|$

k	t_f	$\lvert V_k^C\rvert$	$\frac{r_k}{1000}$	k	t_f	$\lvert V_k^C\rvert$	$\frac{r_k}{1000}$
1	$[0,1]$	95583	95583	6	$[2,4]$	16412	32824
2	$[1,2]$	71007	71007	7	$[4,5]$	31887	31887
3	$[0,2]$	30832	61664	8	$[5,6]$	27068	27068
4	$[2,3]$	51363	51363	9	$[0,4]$	5964	23856
5	$[3,4]$	39132	39132	10	$[6,7]$	23594	23594

Listing 2: Structure of the optimized code for the heat equation

```
template <typename T>
void heat_sig( T& t, vector<T>& parameter ) {
  if ( t >= 0 && t <= 1) {
    heat_1(t, parameter); // Code for t in [0,1]
  } else if ( t >= 1 && t <= 2) {
    heat_2(t, parameter); // Code for t in [1,2]
  } else if ( t >= 2 && t <= 3) {
        .
        .
        .
  } else {
    heat(t, paramter); // Original code
  }
}
```

represent the different section of the code they belong to. The black framed nodes are the user registered variables while the red framed nodes are insignificant for the computation of the temperature at the center of the rod. We observed that for small t_f there is only a small range around the center of the rod influencing the temperature at this position such that only those computations are significant for the results. For larger values of the simulation time t_f the range of influence expands and more computations become significant.

In Tab. 1 the top ten of the 63 generated scenarios are listed with their range for the final time t_f, their number of insignificant computations and their calculated rank. The rank r_k is divided by the width of the input domain size for the initial temperature $w([T(0,x,c)]) = 20$ and the temperate at the right end $w([T(t,2,c)]) = 50$ because they are unchanged. For the best ranked scenario s_1 with an input domain of $t_f \in [0,1]$ almost 60% of the computations are marked as insignificant, while scenario s_2 has just 44% insignificant computations.

The insignificant computations V_k^C for each scenario can be used to obtain special code variants. We observe that the third ranked scenario s_3 with input domain $t_f \in [0,2]$ is already covered by the input domains of the first two scenarios s_1 and s_2 such that there is no need to generate code for this scenario. In general, if $D_k \in \bigcup_{i=1}^{k} D_i$ there is no need to implement scenario s_k. A resulting sketch of the code is given in Listing 2 in which the functions `heat_k()` executes only the significant computations of scenario s_k.

IV. Summary and Outlook

We presented IASA and introduced the interval splitting approach. A case study of a simulation of the 1-D heat equation was analyzed and the results were used as a sanity check

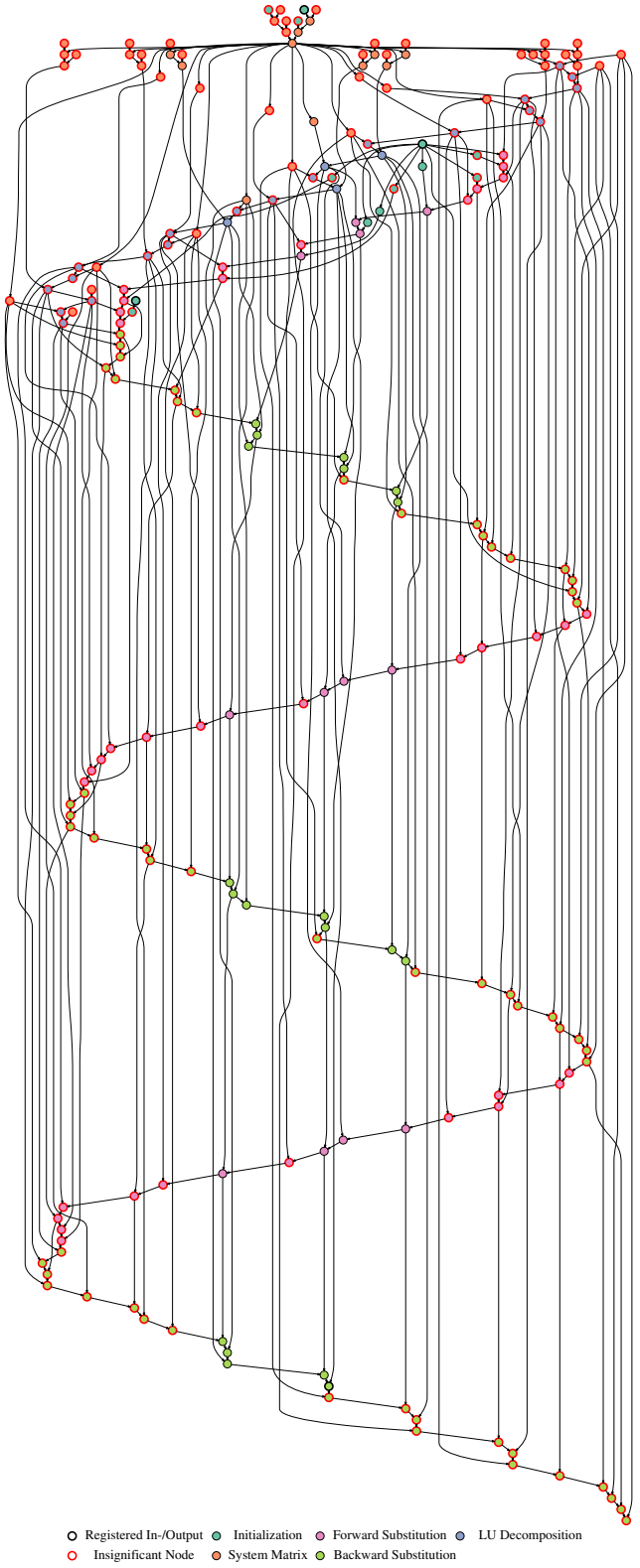

Fig. 3: Automatically generated advanced significance DAG G^{S+} for the simulation of the heat equation with $m = 3$ time steps and $n = 9$ space distributions on the rod to illustrate the structure of the code and which nodes are insignificant

for IASA. The significance tool recognizes the sparsity of the system matrix and indicates that the dense solver should be replaced by a sparse one. Furthermore, the quantification mode was used to verify that simulations with a short simulation time have a restricted range of influence that expands for larger simulation times. By replacing computations with constants IASA can save up to 60% of the total computations obtained with the sparse solver for special scenarios.

IASA is currently limited to continuous differentiable functions. Potential problems has to be investigated arisen from the naive interval evaluation of a function by simply replacing floating point operations with interval operation. Part of the future work is to handle comparisons of two intervals that intersect and find a heuristic which variables should be split. We will implement the interval splitting with the exploration mode to apply IASA on larger computer codes without additional knowledge and automatically generate more efficient code variants of black box code.

REFERENCES

[1] D. Mohapatra, G. Karakonstantis, and K. Roy, "Significance driven computation: a voltage-scalable, variation-aware, quality-tuning motion estimator," in *Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design*. ACM, 2009, pp. 195–200.

[2] S. Misailovic, M. Carbin, S. Achour, Z. Qi, and M. C. Rinard, "Chisel: reliability-and accuracy-aware optimization of approximate computational kernels," in *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*. ACM, 2014, pp. 309–328.

[3] J. Riehme and U. Naumann, "Significance analysis for numerical models," *WAPCO*, 2015. [Online]. Available: http://wapco.inf.uth.gr/2015/papers/SESSION3/WAPCO_3_1.pdf

[4] J. Deussen, J. Riehme, and U. Naumann, "Automation of significance analyses with interval splitting," 2015.

[5] V. Vassiliadis, J. Riehme, J. Deussen, K. Parasyris, C. D. Antonopoulos, N. Bellas, S. Lalis, and U. Naumann, "Towards automatic significance analysis for approximate computing," cGO.

[6] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to interval analysis*. SIAM, 2009.

[7] A. Griewank and A. Walther, "Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation," *ACM Transactions on Mathematical Software (TOMS)*, vol. 26, no. 1, pp. 19–45, 2000.

[8] ——, *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[9] U. Naumann, *The art of differentiating computer programs: an introduction to algorithmic differentiation*. SIAM, 2012, vol. 24.

[10] A. Neumaier, *The wrapping effect, ellipsoid arithmetic, stability and confidence regions*. Springer, 1993.

[11] J. Riehme and U. Naumann, "D1.1: Significance Based Computing Modeling," RWTH Aachen, Tech. Rep., June 2014. [Online]. Available: www.scorpio-project.eu/wp-content/uploads/2014/07/Scorpio_D1.1.pdf

[12] J. Lotz, K. Leppkes, and U. Naumann, "dco/c++ - derivative code by overloading in c++," RWTH Aachen, Tech. Rep. AIB-2011-06, May 2011. [Online]. Available: http://aib.informatik.rwth-aachen.de/2011/2011-06.ps.gz

[13] M. Lerch, G. Tischler, J. W. von Gudenberg, W. Hofschuster, and W. Krämer. `FILIB++` interval library. [Online]. Available: www2.math.uni-wuppertal.de/~xsc/software/filib.html