2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)

A Research-Based Course Module to Study Non-determinism in High Performance Applications

Patrick Bell
University of Tennessee Knoxville
Knoxville, USA

Kae Suarez
University of Tennessee Knoxville
Knoxville, USA

Barbara Fossum
University of Tennessee Knoxville
Knoxville, USA

Dylan Chapp
University of Tennessee Knoxville
Knoxville, USA

Sanjukta Bhowmick
University of North Texas
Denton, USA

Michela Taufer
University of Tennessee Knoxville
Knoxville, USA

***Abstract*—We present a research-based course module to teach computer science students, software developers, and scientists the effects of non-determinism on high performance applications. The course module uses the ANACIN-X software package, a suite of software modules developed by the authors; ANACIN-X provides test cases, analytic tools to run different scenarios (e.g., using different numbers of processes and different communication patterns), and visualization tools for beginner, intermediate, and advanced level understandings in non-determinism. Through our course module, students in computer science, software developers, and scientists gain an understanding of non-determinism, how to measure its occurrence in an execution, and how to identify its root causes within an application's code.**

***Index Terms*—High Performance Computing, Parallelism, Message Passing, Workforce Development, Debugging**

I. Introduction

Programming techniques based on message passing such as MPI have become crucial to the scalable implementation of scientific codes using high performance computing (HPC) [1]. Achieving high scalability requires asynchronous execution of scientific codes, but such asynchronous executions often exhibit non-deterministic behaviour (i.e., multiple runs of the same code produce different patterns of execution). This non-determinism can hinder the efforts of software developers and scientists to identify bugs in their code or to assess the correctness of their results [2]–[5]. For example, in the Enzo software package for automatic identification of galactic halos [3], [6], different galactic halos were identified across different runs due to the non-deterministic order of message exchanges. It is thus essential for students in computer science, software developers, and scientists to gain an understanding of non-determinism, along with how to mitigate the issues when they arise.

Tools have been developed to help researchers address the challenges posed by non-deterministic executions [7]–[9]. But as of yet, there is no interactive system for introducing users to understanding and addressing non-determinism. We fill this gap by developing a research-based course module that builds on the ANACIN-X software package and its environment. ANACIN-X was developed by the authors as part of their research on scientific applications exhibiting different outcomes across multiple runs on large-scale systems [10], [11]. In [10], the ANACIN-X software package is described in detail. In [11], ANACIN-X is used to evaluate and understand the non-determinism present in two mini-applications: miniAMR [12] and the Monte Carlo Benchmark (MCB) [13]. In this course module, the ANACIN-X environment enables users to mimic the execution of Message Passing Interface (MPI) applications on HPC platforms.

Our course module focuses on teaching methods to understand the causes and effects of non-determinism in HPC applications using the message passing paradigm. The module can be included into a larger course on parallel programming using programming languages such as MPI or used as a half-day tutorial at conferences. The course module is oriented toward undergraduate and graduate students in computer science, software developers of HPC codes, and scientists working in HPC. Often, undergraduate and graduate degree programs in computer science introduce students to the concept of parallel code, but leave a more in-depth understanding of the code executions on large scale HPC systems up to individual exploration. Many software developers and scientists working in HPC have a familiarity with parallel programming but have not been taught how to recognize and address non-determinism when it arises. This paper and the associated course module will therefore:

- Introduce students, software developers, and scientists to non-determinism and its impact on code executions;
- Define functionalities of the ANACIN-X tool, specifically how to access the suite of modules via GitHub, how to use it to understand and manage non-determinism for multiple mini-applications, and how to apply it to a broad set of applications; and
- Demonstrate the use of ANACIN-X through its visualization and analysis tools and the study of non-determinism through the visualizations.

In this paper, Section II defines the course module structure and prerequisites, the ANACIN-X software, and the com-

This work is supported by the National Science Foundation CCF #1900888 and #1900765.

978-1-6654-9747-3/22/$31.00 ©2022 IEEE
DOI 10.1109/IPDPSW55747.2022.00067

putational requirements. Section III presents three use cases that are supported by the ANACIN-X environment and the lessons that can be learned from them. Section IV provides a short overview of related work, and Section V presents our conclusions.

II. Course Module Outline

A. Course Module Structure and Prerequisites

Our course module has three levels of complexity: beginner, intermediate, and advanced. The course module is problem-driven: topics at each level are addressed through use cases in which we define one or multiple problems and students implement their solutions. Each level explains different topics, targets different objectives, and has different prerequisites. Table I shows the learning objectives at each level of complexity.

Students should have the prior knowledge for each level of difficulty presented in Table II before using this module:

In addition to the listed prerequisites, students must know or learn terminology relevant to the course. Throughout the course, we use the terms *event graph*, *kernel*, and *root source of non-determinism*. An event graph refers to a graph model of the MPI communication pattern of a given application. Nodes of an event graph correspond to MPI function calls and edges correspond to on-process or inter-process communication. Figure 1 displays a visualization of what an event graph can look like. Event graphs encode time by treating on-process communication as logically ordered (i.e., logical time). A kernel function (also called the graph kernal) is a similarity function that can be selected from many possible graph kernel functions. Formally, a graph kernel is an inner product function across the embeddings of two graphs in a Reproducing Kernel Hilbert Space [14]. Students should be taught that the kernel functions as a scalar-valued measure of the similarity between two graphs. For more details on graph kernels in general, see the recent survey [15]. In this module, we apply the kernel distance to event graphs, which is calculated directly from a kernel and measures the difference between the graphs. Because event graphs correspond to the communication pattern of an application, the kernel distance corresponds to the calculated difference between two runs of that application, and thus serves as a proxy metric for non-determinism. A root source of non-determinism refers to a function or set of functions in the source code of an application that can cause non-deterministic executions.

B. Software and Platform

We use the ANACIN-X software package to generate use cases and their visualizations; we leverage the use cases to teach concepts in the course. ANACIN-X is a modular framework for automated analysis of non-deterministic MPI applications. ANACIN-X provides the following features that make it useful to the course module:

1) Generation of event graph models communication patterns.
2) Measurement of the amount of non-determinism within an application at varying settings.
3) Automatic identification of root causes of non-determinism.
4) Visualization of communication patterns, amounts of non-determinism, and non-determinism sources.

ANACIN-X measures non-determinism in executions through a proxy metric such as the kernel distance between two event graphs representing two distinct executions. For a full description of the software and its settings and configurations, see the software GitHub page [16] and research work of the authors [10], [11].

Instructors can use the course module on their local clusters or on one of two open-access, free of charge platforms: Code Ocean [17] and Jetstream [18]. The GitHub version of ANACIN-X [16] contains all the software and information to install it on local clusters. A version of the ANACIN-X environment is installed and made available to the community on both platforms. Code Ocean is a platform for reproducibility of software via version control of software and pre-built cloud workstation environments. We created a training environment that can be directly accessed through the Code Ocean online portal [17]. Jetstream is a cloud-based computing environment on which users can generate virtual machines to develop and run software. We created a training image containing all of the dependencies for ANACIN-X that can be directly searched via the tag `ANACIN-X` once one logs into the XSEDE platform.

Instructors can use these features of ANACIN-X on all three platforms:

- Number of MPI processes: Determines how many MPI processes a communication pattern is generated across. This feature is used for learning levels A, B, and C.
- Percentage of non-determinism: Determines how non-deterministic a communication pattern is as a percentage from 0% up to 100%. This feature is used for learning levels A, B, and C.
- Number of compute nodes: Determines the number of separate compute nodes a communication pattern is generated across. This feature is used for learning levels A, B, and C.
- Number of communication pattern iterations: Determines how many times a communication pattern appears during an execution. This feature is used for learning levels B and C.

The current version of ANACIN-X on Code Ocean and Jetstream does not support multiple compute nodes because of the platform constraints and the image setting respectively. Multiple MPI processes can still be used on either system.

ANACIN-X comes packaged with three MPI-based mini-applications with different communication patterns: (a) message race, (b) Algebraic Multigrid (AMG) 2013 [19], and (c) Unstructured Mesh [20]. A detailed understanding of these communication patterns is not needed prior to starting the course. Instructors should emphasize to their students how the three mini-applications each have varying complexity in

A. Beginner level	•Goal A.1: Introduce parallelism using the message passing paradigm •Goal A.2: Define non-determinism associated to message passing
B. Intermediate level	•Goal B.1: Study effects of number of processes on non-determinism in applications •Goal B.2: Study non-determinism across multiple iterations of the same code during the same application execution
C. Advanced level	•Goal C.1: Quantify the level of non-determinism in application's executions •Goal C.1: Identify root sources of non-determinism in applications

TABLE I: An outline of the learning objectives for each level of difficulty in this course module.

A. Beginner level	•A basic knowledge of MPI, in particular point-to-point MPI communication calls. •A basic knowledge of graph theory, but not necessarily an in-depth understanding.
B. Intermediate level	•An understanding of non-determinism from the topics described by the beginner level. •The ability to interpret violin plots.
C. Advanced level	•An understanding of what external factors impact the amount of non-determinism in an application from the intermediate level. •The ability to understand C++ source code to identify functions causing non-determinism.

TABLE II: An outline of the prerequisite knowledge for each level of difficulty in this course module.

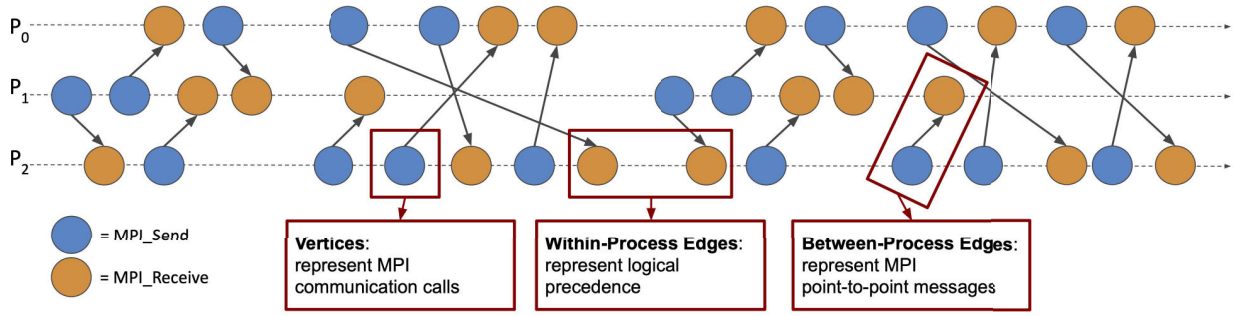


Fig. 1: An example of an event graph representation of an MPI communication pattern between three MPI processes. Nodes of the graph represent MPI events (i.e., MPI_Send() and MPI_Recv()). Edges within a process represent logical precedence of MPI events. Edges between processes represent point-to-point MPI messages.

their source code and represent well-known communication patters in real applications [21]. Specifically, a message race is when multiple messages are being sent to the same process, and the order they will arrive in is unknown. It is the simplest communication pattern of the three. AMG 2013 expands on the message race pattern by allowing each process to send a message to all other processes. Each process in an AMG 2013 pattern does this twice. Unstructured Mesh expands further by randomizing which processes are allowed to communicate with each other. Students can run tests on each mini-application to compare how the application's complexity affects the amount of non-determinism and to identify code functions that act as root sources of non-determinism (i.e., call-paths of MPI functions). See the research work of the authors [11] for more details about each of the three communication patterns.

ANACIN-X provides users with these tools for visualization:

- Event graph visualization: Displays the event graph which corresponds to the communication pattern being executed.
- Kernel distance visualization: A violin plot of the sample of kernel distances calculated for the input MPI application.
- Callstack visualization: A bar plot of the relative frequency of call-paths of MPI functions during periods of highly non-deterministic execution across the logical time of an event graph.

The kernel distance visualization and the callstack visualization can also be generated via a Jupyter Notebook packaged with the software in the ANACIN-X GitHub.

C. Computational Requirements

The ANACIN-X software in GitHub is public under Apache License 2.0. Students and instructors using Code Ocean for the training need to create an account on the platform. Accounts are free and should provide enough computational units to run the ANACIN-X environment. Similarly, students and instructors using Jetstream must have an XSEDE account. Instructors must acquire an educational allocation for the Jetstream cloud. Once logged into Jetstream, students and instructors should use the image that can be searched with the tag `ANACIN-X`.

When the training takes place on local HPC resources, there are several requirements:

- The ability to clone the GitHub repository with the software package, compile it, and run it on the local cluster.
- Access to a computer with multiple processes. The largest use cases referenced in this paper ran on a 32-process cluster, but smaller clusters can generate similar results.

- Access to a computer with at least two compute nodes.

III. Use Cases to Understand Non-Determinism

A. Use Case 1: Distributed Computing and Non-determinism

Students first gain an understanding of both message passing and communication non-determinism. Specifically, students at a beginner level should be able to answer the following questions once they have completed Use Case 1:

- What is message passing in the context of an execution (Goal A.1)
- What is non-determinism in the context of an execution (Goal A.2)

When introducing the concepts of distributed computing and non-determinism, it is very helpful to visualize the communication patterns of small applications. By doing so, students can see what message passing and non-deterministic executions look like.

1) Goal A.1: Introduce parallelism using the message passing paradigm: For students to satisfy Goal A.1, they must understand two key concepts: (1) any process in an execution can send a message to another process and (2) processes can exchange messages using different communication patterns. To develop the understanding of the first concept, students are presented with multiple event graph visualizations corresponding to different scenarios. For example, in Figure 2, multiple processes send independent messages to a single receiving process; in Figure 3, one process sends one message to another process while receiving a second message from the other process asynchronously. Each figure is generated by ANACIN-X. In the figures, each row corresponds to a different MPI rank (process). Green circles correspond to the start or end of a process. Blue circles correspond to sending a message, and red circles correspond to receiving a message. The students are asked to describe the scenarios in each figure.

To develop the understanding of the second concept, students are presented with the same set of figures but the emphasis is put on the different communication patterns. Figure 2, for example, has three MPI processes, each sending a message to the fourth process. This is an event graph visualization of a message race communication pattern. Figure 3, on the other hand, has two MPI processes each send a message to the other. Afterward, each process sends a message back asynchronously. This is a small-scale replication of the AMG 2013 communication pattern. The students are asked to reproduce the scenarios in the two figures and further expand the type and number of scenarios by running ANACIN-X with any number of processes and selecting among the different communication patterns supported by ANACIN-X.

2) Goal A.2: Define non-determinism associated to message passing: To satisfy Goal A.2, students must be able to understand that non-determinism is defined by inconsistent type and order of messages exchanged between MPI processes across runs of the same application execution. Therefore students are first shown scenarios depicted in figures such as Figure 4a and Figure 4b and then are asked to generate new scenarios

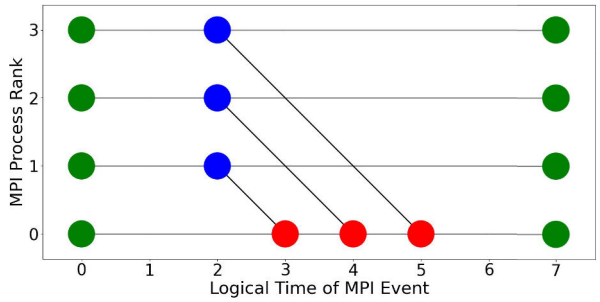


Fig. 2: Event graph visualization of a message race communication pattern on four MPI processes. Each row corresponds to a different MPI rank (process). Green circles correspond to the start or end of a process; blue circles correspond to sending a message; and red circles correspond to receiving a message.

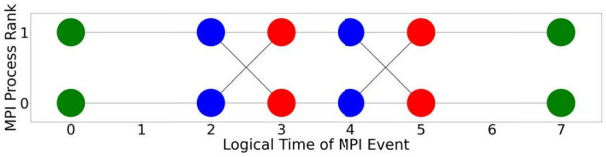
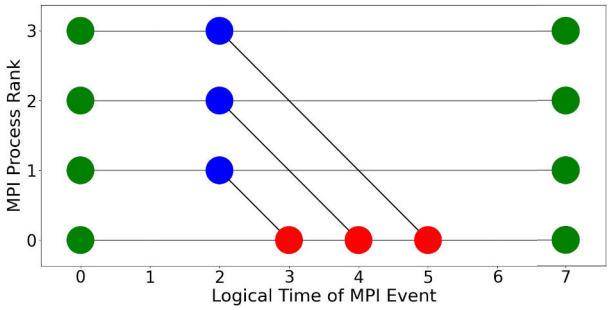


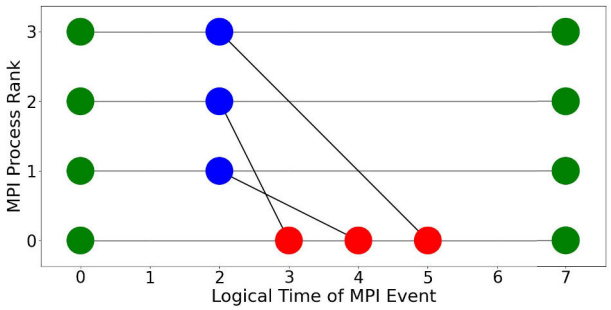
Fig. 3: Event graph visualization of AMG 2013 communication pattern on 2 MPI processes. Each row corresponds to a different MPI rank (process). Green circles correspond to the start or end of a process; blue circles correspond to sending a message; and red circles correspond to receiving a message.

using ANACIN-X. Both Figures 4a and Figure 4b are event graphs like those in Figures 2 and 3. They both correspond to the message race communication pattern. Instructors should stress that both Figure 4a and Figure 4b are generated using the same code with the same inputs but are the result of two independent runs.

Students can observe that, despite being executed using the same code and the same inputs, the runs from Figure 4a and Figure 4b have different communication patterns. Specifically, the messages sent by Process 1 and Process 2 do not arrive at Process 0 in the same order. Students use this example and other examples generated by ANACIN-X to understand that non-determinism is when multiple executions of the same code, run in the same way, produce different communication patterns. The differences in communication patterns are possible because the ANACIN-X environment is set to mimic delays in individual messages due to network and I/O congestion, or CPU contention, for example. Students can define an average degree of delay by setting the environment's percentage of non-determinism. Figure 4a and Figure 4b were generated by running ANACIN-X with four processes and 100% non-determinism. Tests to reproduce scenarios such as those in Figures 4a and 4b should be run across multiple compute nodes to increase the likelihood that runs are non-



(a)



(b)

Fig. 4: Two event graph visualizations run on the same configuration of the message race communication pattern. Different graphs correspond to two different non-deterministic executions. Each row corresponds to a different MPI rank (process). Green circles correspond to the start or end of a process; blue circles correspond to sending a message; and red circles correspond to receiving a message.

deterministic.

B. Use Case 2: Factors that Impact Non-determinism

Non-determinism can be difficult to reproduce [22]. In such situations, developers and scientists need to know what factors impact the amount of non-determinism present to make the non-determinism more or less significant. For this reason, intermediate students should be able to identify the factors. Specifically, they should be able to answer the following questions:

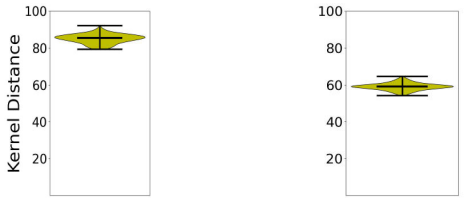
- What is the effect of increasing the number of MPI processes used during execution? (Goal B.1)
- What is the effect of increasing the number of communication pattern iterations? (Goal B.2)

ANACIN-X provides the tools to explore these topics. Specifically, students can take the following steps to investigate the effects of different settings:

- Run the ANACIN-X software on the mini-applications provided with the software to gather data on a setting.
- Adjust the simulation settings to change either the number of MPI processes or the number of communication pattern iterations.
- Run the same application many times to collect a sample of non-deterministic executions.
- Visualize the amount of non-determinism across the executions to compare the amount of non-determinism across the different settings.

We demonstrate how using ANACIN-X with these steps teaches students about the factors that impact non-determinism.

1) Goal B.1: Study Effects of Number of Processes: To satisfy Goal B.1, students must know that the number of MPI processes in a simulation is directly related with the amount of non-determinism in the simulation: increasing the number of processes will increase the amount of non-determinism, and vice versa. To this end, students are shown figures of the same type as Figure 5a and Figure 5b. Each figure is a violin plot across 20 data points that displays the measured amount of non-determinism within the communication pattern at the setting along the X-axis. We use the Unstructured Mesh communication pattern for these two figures, but other patterns are available in ANACIN-X. The simulation used to generate Figure 5a was run on 32 MPI processes; the simulation used in Figure 5b was generated in the same way as Figure 5a but by changing the number of MPI processes to 16.



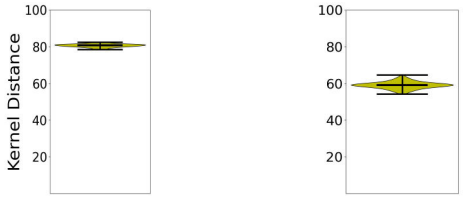
(a) 32 MPI Processes (b) 16 MPI Processes

Fig. 5: Kernel distances for 20 executions of the Unstructured Mesh mini-application. The executions in (a) are performed on 32 MPI processes; and the executions in (b) are performed on 16 MPI processes. The kernel distance is a proxy for non-determinism: the higher, the more non-deterministic the execution.

Students will use Figures like Figure 5 to observe that the number of processes and the amount of non-determinism are directly related in the Unstructured Mesh simulation. ANACIN-X should be run with 100% non-determinism to reproduce these results. Similar results can be produced using the other two communication pattern benchmarks packaged with ANACIN-X. As an assignment, students should run ANACIN-X with similar settings on the other benchmarks to gain more familiarity with this concept. By understanding this concept, the student will know to increase the number of processes in their simulations to help reveal non-determinism when it is difficult to reproduce.

2) Goal B.2: Study Non-determinism across Iterations: Scientific applications often consist of the iterative execution of the same software code. These iterations during the same application execution exhibit non-determinism that is easy to

identify. To satisfy Goal B.2, students should be able to link the iterations of an application execution to the amount of non-determinism in the application. Students are shown a set of figures of the same type as Figures 6a and 6b, which capture this property. The Unstructured Mesh benchmark application is used to generate the data from both of these two figures. The applications used to generate the data in each figure were run on 16 MPI processes. Figure 6a was generated by executing the same Unstructured Mesh pattern with two iterations across those 16 processes, whereas Figure 6b was generated by only executing the Unstructured Mesh pattern with one iteration across the processes. We generate 20 data points (i.e., application runs) for each figure to improve the statistical significance of the results. Students can use figures



(a) 2 Iterations (b) 1 Iteration

Fig. 6: Kernel distances for 20 executions of the Unstructured Mesh mini-application. The executions in (a) are performed with two iterations of the core application code; and the executions in (b) are performed with single iterations of the core application code. The kernel distance is a proxy for non-determinism: the higher, the more non-deterministic the execution.

such as Figure 6 to discover if there is a link between iterations and amount of non-determinism in their application. For this test, ANACIN-X should be run with 100% non-determinism to produce results with significant non-determinism across iterations. These tests enable students to understand how, in applications exhibiting non-determinism, by increasing the number of iterations in their application, they may accumulate substantial differences in the numerical results and ultimately different scientific findings, as shown in [3] .

C. Use Case 3: Root Sources of Non-determinism

Last, a course on non-determinism should train students to identify potential root sources of non-determinism in their code: functions within code that can produce non-deterministic communication patterns. To this end, advanced students should be able to answer the following questions at an advanced level of understanding in non-determinism:

- How do root sources of non-determinism impact the amount of non-determinism? (Goal C.1)
- How can ANACIN-X be used to identify root sources of non-determinism? (Goal C.2)

For each of the advanced level goals, we show approaches to answer the questions by using the AMG2013 mini-application and its communication patterns. Students should continue their exploration of the topics by demonstrating their understanding of each educational objective on the other mini-application communication patterns packaged with ANACIN-X and further extend the approach for other applications.

1) Goal C.1: Quantify Amount of Non-determinism in Executions: Root sources of non-determinism are embedded in an application's code. The access to an application's code is not always guaranteed; when available, the code may be very complex to parse and understand. To demonstrate understanding of Goal C.1, students should be able to explain that root sources of non-determinism can be used to directly control the amount of non-determinism in an application. Figure 7 is an example of results from running the AMG 2013 non-deterministic mini-application in the ANACIN-X environment, without parsing the code. The X-axis corresponds to the actual percentage of non-determinism present within the application. The Y-axis corresponds to the measured (un-normalized) amount of non-determinism. The actual percentage of non-determinism is defined during communication pattern generation as the percentage of messages that can suffer from congestion or contention delays and thus exhibit a non-deterministic arrival pattern. For the benchmark patterns packaged within ANACIN-X, the ability to set the percentage of non-determinism is provided as an option to users through the inputs of ANACIN-X. The measured amount of non-determinism corresponds to the kernel distance across executions. We collect 20 data points (i.e., application runs) for each percentage of non-determinism to improve the statistical significance of the measurements. Using figures such as Figure 7, students can quantify the amount of non-determinism within one execution and demonstrate whether such an amount is directly related to the actual percentage of non-determinism for a given application. That is, by changing the percentage of non-determinism present in the usage of the code's functions (i.e., the root source of non-determinism), one can directly control the amount of non-determinism present in the execution of the code. The results in Figure 7 were generated using 32 MPI processes and percentages of non-determinism ranging from 0% up to 100% at intervals of 10%. In addition, only one compute node and one communication pattern iteration were used to generate the results.

2) Goal C.2: Use ANACIN-X to Identify Root Sources of Non-determinism: It is not straightforward to control the amount of non-determinism within an application when the root source of that non-determinism is not already known and is embedded in the called system functions. For this reason, students should learn how to identify likely root sources of non-determinism in their code. We propose an example in which we use ANACIN-X to identify non-determinism root sources within executions such as the AMG2013 mini-application in Figure 8. The ANACIN-X environment identifies the callstacks in the application and measures their frequency. In the figure, the X-axis corresponds to the list of callstacks in the application's run that were identified as taking place during high periods of non-determinism. The Y-axis corresponds to the normalized relative frequency of the

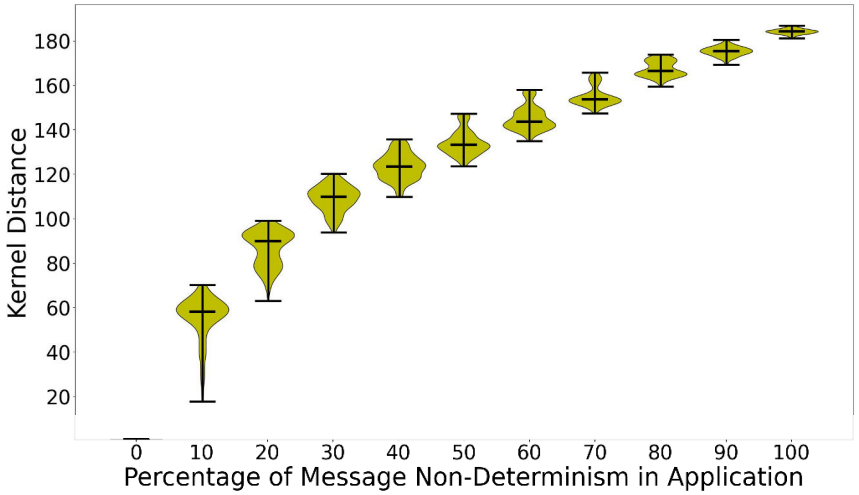


Fig. 7: Kernel distance visualization of the AMG2013 mini-application and its communication pattern on 32 MPI processes. We vary the percentage of non-determinism from 0% up to 100% at increments of 10%. Simulation is made using 1 compute node, 1 communication pattern iteration, and message sizes of 1 byte.

identified callstacks as they appear in an execution's event graph. Students use the figure to observe which MPI functions were identified in regions of high amounts of non-determinism. These functions are likely root sources of non-determinism because MPI functions that take place during periods of high non-determinism are likely related to the non-determinism. Students can extend the ANACIN-X environment to support their own application: to determine where in their code they should look to manage their non-determinism by locating the callstack functions associated to the high non-determinism. Using the skills gained by identifying root sources of non-determinism, students can understand the non-determinism within their own codes.

IV. Related Work

A recent report [2] highlights the need for developers and scientists to gain understanding and control of non-determinism when identifying non-deterministic bugs in HPC software. One debugging case study [22] reported more than 10,000 hours of compute time spent manually locating a non-deterministic bug in version 2.10.1 of the HYPRE linear algebra package [23]. Another study on replicability of results in computational science [3] demonstrates inconsistent results across multiple runs of the Enzo astrophysics software [6] due to non-determinism.

In addition to ANACIN-X, some other tools have been developed to help developers understand and control non-determinism. Record and replay tools [7] like ReMPI [24] suppress non-determinism to temporarily improve reproducibility of results. Tools for crash detection like PopMine [9] identify causes of software crashes, including those in non-deterministic applications, but it is ineffective when the non-deterministic bug does not cause a software crash. Tools for motif detection like SABALAN [8] learn a model of communication pattern motifs that produce non-determinism. Here, we use the tools within ANACIN-X because it can evaluate root sources of non-determinism in non-crashing applications without being limited by motifs and because it can visualize multiple aspects of non-determinism.

V. Conclusion

The ANACIN-X software package is used to design a research-based course module on three levels of understanding in non-determinism: (1) what non-determinism is; (2) what factors impact the amount of non-determinism in an application; and (3) how to identify the root causes of non-determinism. We select ANACIN-X as the software to teach these topics because it can produce visualizations of communication pattern event graphs, measurements of the amount of non-determinism in an application, and functions within code that likely have an impact on the amount of non-determinism. ANACIN-X can be used to teach undergraduate and graduate students in computer science, software developers of HPC codes, and scientists working in HPC; the course module can be used as part of a parallel computing course or as a half-day tutorial. We will extend ANACIN-X in future work to support communication patterns beyond the one-to-one MPI communication calls currently supported, including collective MPI operations.

Acknowledgment

This work was supported in part by the National Science Foundation under Grant 1900888 and Grant 1900765. The authors acknowledge IBM through a Shared University Research Award. Results generated in part with support of the Tellico

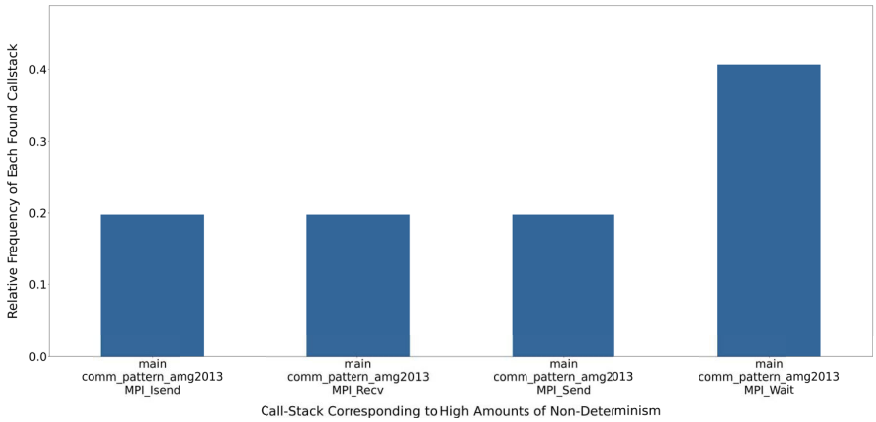


Fig. 8: Callstack visualization for the AMG2013 mini-application and its communication pattern. The settings are the same as in Figure 7

cluster computer and the XSEDE computational resources Stampede2 cluster computer and Jetstream cloud computer.

REFERENCES

[1] I. Laguna, R. Marshall, K. Mohror, M. Ruefenacht, A. Skjellum, and N. Sultana, "A Large-scale Study of MPI Usage in Open-source HPC Applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–14.

[2] G. Gopalakrishnan, P. D. Hovland, C. Iancu, S. Krishnamoorthy, I. Laguna, R. A. Lethin, K. Sen, S. F. Siegel, and A. Solar-Lezama, "Report of the HPC Correctness Summit, Jan 25–26, 2017, Washington, DC," *arXiv preprint arXiv:1705.07478*, 2017.

[3] V. Stodden and M. S. Krafczyk, "Assessing Reproducibility: An Astrophysical Example of Computational Uncertainty in the HPC Context," in *Proceedings of the 1st Workshop on Reproducible, Customizable and Portable Workflows for HPC at SC'18*, 2018.

[4] D. Chapp, T. Johnston, and M. Taufer, "On the Need for Reproducible Numerical Accuracy through Intelligent Runtime Selection of Reduction Algorithms at the Extreme Scale," in *Proceedings of the 2015 IEEE International Conference on Cluster Computing (CLUSTER)*. Chicago, IL, USA: IEEE Computer Society, September 8 – 11 2015, pp. 166–175.

[5] M. Taufer, O. Padron, P. Saponaro, and S. Patel, "Improving Numerical Reproducibility and Stability in Large-Scale Numerical Simulations on GPUs," in *Proceedings of the 24th IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*. Atlanta, Georgia, USA: IEEE Computer Society, 19-23 April 2010, pp. 1–9.

[6] G. L. Bryan, M. L. Norman, B. W. O'Shea, T. Abel, J. H. Wise, M. J. Turk, D. R. Reynolds, D. C. Collins, P. Wang, S. W. Skillman *et al.*, "Enzo: An Adaptive Mesh Refinement Code For astrophysics," *The Astrophysical Journal Supplement Series*, vol. 211, no. 2, p. 19, 2014.

[7] D. Chapp, K. Sato, D. H. Ahn, and M. Taufer, "Record-and-Replay Techniques for HPC Systems: A Survey," *Supercomput. Front. Innov.: Int. J.*, vol. 5, no. 1, p. 11–30, mar 2018. [Online]. Available: https://doi.org/10.14529/jsfi180102

[8] S. Alimadadi, A. Mesbah, and K. Pattabiraman, "Inferring Hierarchical Motifs from Execution Traces," in *Proceedings of the 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 776–787.

[9] E. Seo, M. M. H. Khan, P. Mohapatra, J. Han, and T. F. Abdelzaher, "Exposing Complex Bug-Triggering Conditions in Distributed Systems via Graph Mining," in *Proceedings of the International Conference on Parallel Processing, ICPP 2011, Taipei, Taiwan, September 13-16, 2011*, 2011, pp. 186–195.

[10] P. Bell, K. Suarez, D. Chapp, N. Tan, S. Bhowmick, and M. Taufer, "ANACIN-X: A Software Framework for Studying Non-determinism in MPI Applications," *Software Impacts*, vol. 10, p. 100151, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2665963821000634

[11] D. Chapp, N. Tan, S. Bhowmick, and M. Taufer, "Identifying Degree and Sources of Non-Determinism in MPI Applications Via Graph Kernels," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 12, pp. 2936–2952, 2021.

[12] C. T. Vaughan and R. F. Barrett, "Enabling Tractable Exploration of the Performance of Adaptive Mesh Refinement," in *Proceedings of 2015 IEEE International Conference on Cluster Computing*. IEEE, 2015, pp. 746–752.

[13] N. Gentile and B. Miller, "Monte Carlo Benchmark (MCB)," https://computing.llnl.gov/projects/co-design/mcb, LLNL-CODE-507091.

[14] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph Kernels," *Journal of Machine Learning Research*, vol. 11, no. Apr, pp. 1201–1242, 2010.

[15] N. M. Kriege, F. D. Johansson, and C. Morris, "A Survey on Graph Kernels," *Applied Network Science*, vol. 5, no. 1, pp. 1–42, 2020.

[16] D. Chapp, P. Bell, K. Suarez, S. Bhowmick, and M. Taufer, "ANACIN-X: Analysis and Modeling of Nondeterminism and Associated Costs in eXtreme Scale Applications," https://github.com/TauferLab/ANACIN-X,.

[17] P. Bell, D. Chapp, K. Suarez, N. Tan, S. Bhowmick, and M. Taufer, "ANACIN-X: Analysis and modeling of Nondeterminism and Associated Costs in eXtreme scale applications," https://codeocean.com/capsule/0309009/tree/v4.

[18] N. Tan, P. Bell, S. Bhowmick, and M. Taufer, "Ubuntu20.04_Anacin-X," https://use.jetstream-cloud.org/application/images/1056.

[19] J. Park, M. Smelyanskiy, U. M. Yang, D. Mudigere, and P. Dubey, "High-performance Algebraic Multigrid Solver Optimized for Multi-core based Distributed Parallel Systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015, pp. 1–12.

[20] N. Jain and A. Bhatele, "Chatterbug Communication Proxy Applications Suite," https://github.com/LLNL/chatterbug, LLNL-CODE-756471.

[21] F. Cappello, A. Guermouche, and M. Snir, "On Communication Determinism in Parallel HPC Applications," in *Proceedings of 19th International Conference on Computer Communications and Networks 2010*. IEEE, 2010, pp. 1–8.

[22] K. Sato, D. H. Ahn, I. Laguna, G. L. Lee, M. Schulz, and C. M. Chambreau, "Noise Injection Techniques to Expose Subtle and Unintended Message Races," in *Proceedings of the 22Nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2017, pp. 89–101.

[23] R. D. Falgout and U. M. Yang, "HYPRE: A library of high performance preconditioners," in *Proceedings of the International Conference on Computational Science*. Springer, 2002, pp. 632–641.

[24] K. Sato, D. H. Ahn, I. Laguna, G. L. Lee, and M. Schulz, "Clock Delta Compression for Scalable Order-replay of Non-deterministic Parallel Applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015, pp. 1–12.