30d (Modeling and Simulation)
DOE FY18, Release 2 Phase II SBIR/STTR

Phase II Proposal

VnV: A Self Documenting Testing Framework for In-situ Verification and Validation in High Performance Computing Applications
Phase I Contract Number: DE-SC0018728

Ben O'Neill[1], PI Gregory Watson (Sub-contractor)[2]

[1] RNET Technologies
240 W Elmwood Dr
Dayton, OH 45459-4296
boneill@RNET-Tech.com

[2] Oak Ridge National Laboratory
1 Bethel Valley Road MS6016
Oak Ridge, TN 37831
watsongr@ornl.gov

Contents

1 Significance, Background, and Technical Approach

RNET Technologies Inc. (RNET) in Dayton, OH and Oak Ridge National Laboratory (ORNL) are responding to the 2019 DOE SBIR/STTR Phase II Release 2 (DE-FOA-0001976). This proposal is for a Phase II contract in succession to an initial Phase I contract (Contract #: DE-SC0018728) awarded for DOE SBIR/STTR Topic 30d (Modeling and Simulation). Based on a prototype developed in Phase I, RNET and ORNL are proposing the development of VnV: a self-documenting testing framework for end-user solution verification and validation in high performance computing applications. This proposal will highlight the need for, and the tremendous value of, the proposed tool in high performance numerical simulations. The goal of the proposed Phase II project will be to develop and demonstrate a production quality implementation of the VnV framework that delivers this value across a broad spectrum of numerical applications.

RNET has extensive SBIR experience in high performance computing, including performance optimization of numerical software and libraries, development of fine-grained power monitoring tools for HPC infrastructure, and the development of software usability tools that enhance the user experience. Dr. Watson, our collaborator at ORNL, is an experienced software professional with extensive knowledge of software design, architecture, and engineering practices. He has experience developing parallel debugging software for HPC environments and is the project leader of the open-source Eclipse Parallel Tools Platform (PTP). We believe that our experience developing advanced numerical software for the HPC community combined with Dr. Watson's experience with scientific software engineering uniquely qualifies our team to develop and commercialize the VnV framework.

1.1 Identification and Significance of the Innovation

1.1.1 Identification and Significance

Numerical modeling and simulation (M&S) is almost always more economical than live testing and prototyping; a fact that has seen wide-scale uptake in the R&D life-cycles of products ranging from $10 polycarbonate toys, up to solar panels, airplane wings and nuclear reactors. With access to high performance computational resources at an all time high, and with exascale computing resources on the horizon; the role M&S has in the design pipelines of next generation technologies is only expected to increase. However, numerical simulations are, by definition, an *approximation* to a real world physical system. As such, it is important that this increased reliance on simulated tests is accompanied by a concerted effort to ensure simulations are fit for the intended purpose. As stated in the DoD best practices guide [2], verification and validation (V&V) of a code should be performed when *"...the risk of making an incorrect decision based on a simulation outweighs the time and cost of performing V&V to ensure that a simulation produces results that are sufficiently accurate and reliable."* One only needs to look to the Sleipner platform accident[7, 6], where an offshore oil platform collapsed due to failures in the finite element simulation, to get an idea of the devastating consequences poorly verified numerical simulations can have on business, the environment, and society.

The staples of a rigorous V&V regimen are:

- The development of a detailed V&V plan.
- The implementation of software development best practices (e.g. version control, unit and regression testing, code reviews, etc.).
- Mathematical and algorithmic testing (convergence analysis, mesh refinement studies, method of manufactured solutions, etc.).
- Development of a broad benchmark testing suite.
- Uncertainty quantification and sensitivity analysis.
- Comparison of simulation results with experimental data and results from third party simulations.
- Review of the implementation and results by experts in the field.
- Documentation of the V&V effort.

V&V is a discrete process that cannot account for each and every possibility. This raises issues in the development of general purpose numerical simulation packages because, while it is the simulation software developers responsibility to ensure the product is mathematically correct, it is ultimately the responsibility of the end user to ensure the solution is a suitable representation of their physical model. After all, the direct costs of a design failure (be it time, money, or loss of life) fall squarely on the shoulders of the end-user; any attempt to shift the blame to the developers of simulation library X will certainly fall on deaf ears.

End-user V&V is particularly important in the DOE Nuclear Energy Advanced Modeling and Simulation (NEAMS) program. NEAMS is developing predictive models for the advanced nuclear reactor and fuel cycle systems using leading edge computational methods and high performance computing technologies [4]. The NEAMS group has placed a significant emphasis on V&V [3]; the NEAMS tools have integrated functionality for input validation (through the workbench and MOOSE), mesh refinement and method of manufactured solution analysis (through MOOSE), and uncertainty quantification and sensitivity analysis with DAKOTA (through the workbench). Despite this, the complex nature of the codes combined with the high stakes nature of nuclear reactor design has created a need for tools that automate solution verification for end-user driven simulations.

To address this need, RNET Technologies Inc. (RNET) and Oak Ridge National Laboratory (ORNL) are proposing the development of the VnV framework; a C/C++ software package that facilitates end-user V&V in general purpose numerical simulation packages (e.g., MOOSE, PETSc, libMesh, Fenics, OpenFoam, etc). The framework will promote the development of *explainable* numerical simulations that, in addition to the traditional simulation solution, produces a detailed report outlining how the solution was obtained and why it should be trusted. The VnV framework will provide simple to maintain V&V to create self verifying, self describing, explainable numerical simulations.

1.1.2 Product Overview and Technical Approach

In this proposal, we make the distinction between the *V&V of a simulation package* and the *end-user driven V&V of a solution obtained using a simulation package*. These two processes are not independent; indeed, the V&V of a numerical simulation package almost always includes a set of verified and validated benchmark tests; likewise, the assertion that a package is verified and validated forms the foundation of trust in solutions obtained by end-users. The focus of the Phase II project will be end-user validation; however, the functionality imparted by the framework can be used in the V&V of simulation packages as well.

The VnV framework will facilitate the development of explainable numerical simulations through:

- **In-situ Testing And Analysis:** Unit tests are an effective mechanism for ensuring a function works as expected. However, unit testing is an unavoidably discrete process that cannot cover every possible outcome. This is particularly true for numerical algorithms because even small changes (e.g., input parameters, mesh geometry, etc.) can cause the algorithms to behave unexpectedly (i.e., diverge,

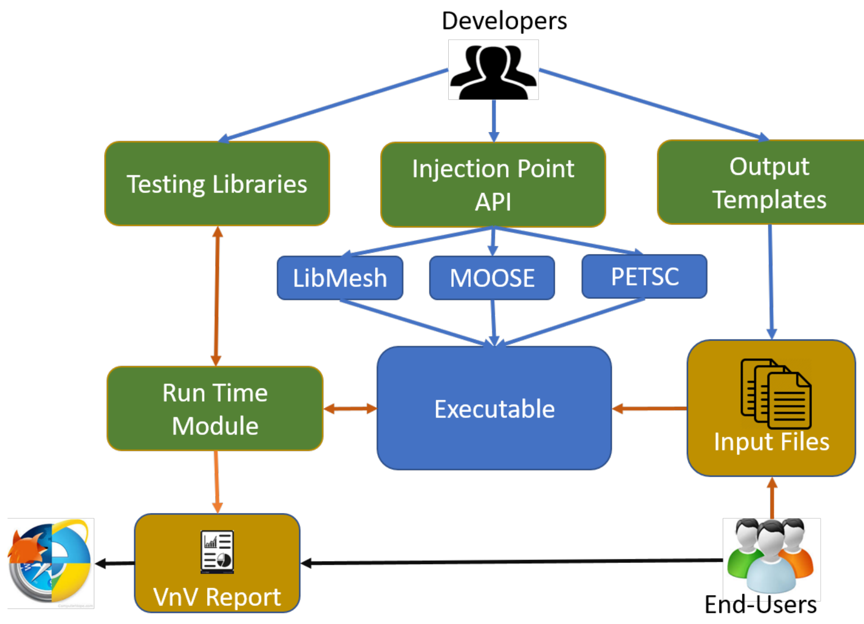


Figure 1: The VnV toolkit. Here, green boxes represent core functionalities. Developer interactions are shown in blue, runtime interactions are shown in orange and post-processing interactions are shown in black.

converge to the wrong solution, etc.). As such, a robust V&V report should include a description of unit tests completed *and* a detailed set of tests and assertions that were completed during the simulation process.

The VnV framework will include a sophisticated test injection system with cross-library support for defining testing points in existing codes. The framework will be able to configure injection points in any library linked to a simulation. For example, a MOOSE user would be able to run V&V tests at injection points defined in the source codes of hypre, PETSc, libMesh, MOOSE, etc through a single interface. This cross library support will allow for in-depth, expert directed, end-user V&V in executables that utilizes a range of numerical simulation libraries.

- **Reusable Software Components:** While the specific details of V&V vary from application to application, the macro scale algorithms are relatively consistent (e.g., mesh refinement studies, the method of manufactured solutions, sensitivity analysis, uncertainty quantification and error propagation). Many of these algorithms can be, or already have been [1], implemented as black-box or near black box solutions. The VnV framework will provide a robust set of near black-box tools that implement these common V&V approaches.
- **Efficiency:** Performing V&V tests in a distributed environment will be expensive, both computationally and due to the data movement required to support generic domain decompositions. The VnV framework will offer functionality for offloading tests to external processes. This will significantly reduce the run time costs of completing V&V using the VnV framework.
- **Documentation Generation:** With software packages under almost constant development, and new and improved packages being released on a regular basis, keeping an up-to-date V&V report is an almost impossible task. The VnV toolkit will include automatic VnV report generation in the form of a server-less HTML web page. The report will be built using an extended markdown format with support for standard markdown formatting, latex formatting, images, videos, self-sorting tables, two-dimensional charts, and three-dimensional visualization.

Figure 1 uses the MOOSE tool-chain to show how developers and end-users will interact with the VnV framework. The first step is to define the injection points. These injection points will be placed at key locations of the code where testing can and should take place. Developers will also complete an output template describing the state of the simulation at each injection point. That specification will be used to populate the final VnV report.

The next step is to create a VnV test. The tests are developed in external libraries and hence, can be developed either by the developer of the simulation or by the end-user of the library. The core framework will also include a robust set of general purpose V&V tests. Each test will be accompanied by a markdown formatted template file. Like injection points, this markdown file will be used to describe the test and present the results. The VnV framework supports a custom markdown format that includes a range of data visualization techniques. We envision that the developers of a numerical simulation package will ship the library with hard-coded injection points and a set of custom V&V tests.

End-users will be able to generate a customized input configuration file for each executable. This configuration file will contain information about every injection point located in the call-graph of the simulation; including those in external 3rd party libraries. After customizing that file, generating a VnV report is as simple as running the simulation.

In summary, once integrated into an application, the VnV framework will provide a simple mechanism for creating self verifying, self describing, explainable numerical simulations. This will significantly reduce the burden associated with V&V for end users, thereby increasing the usability of the tools for non-expert end-users. The core functionality of the VnV toolkit will be released as open source for use in academic and enterprise applications, with RNET providing commercial support, training and integration contracts to interested parties.

1.2 Anticipated Public Benefits

Numerical modeling and simulation (M&S) is extensively used in numerous scientific and engineering fields and disciplines such as computational fluid dynamics, high energy physics, nuclear engineering, and computational finance. The prevalence of smaller scale finite element analysis (FEA) software is even wider, with applications ranging from design optimization in $10 polycarbonate trinkets up to large scale parameter optimization of fighter jet wings. In all cases, the software used to inform the designs of these products must be verified and validated; and in all cases, the verification and validation of this software is a complex, time consuming task requiring input from experts across the broad spectrum of numerical simulation specialty areas (i.e., linear solvers, nonlinear solvers, finite element methods, physical domain specification, data analysis, etc.).

End-User V&V is essential, expensive to setup and maintain, and supporting end-user V&V is difficult. Undetected errors in numerical simulations propagate into physical designs creating issues that can cost millions of dollars to fix, cause catastrophic damage to the environment, and, in the worst case scenarios, result in the loss of human life. While most high quality simulation packages have robust internal V&V regimens, few (if any) ship with a functionality that streamlines end-user V&V . Instead, software simulation developers often take a legal approach, whereby the license includes language to the effect of *use at your own risk* or *this software is released as-is, with no guarantee it is fit for the intended purpose*. The detailed VnV reports do not shift the burden of end-user V&V to the developer, that is, and always will be the responsibility of the end-user; rather, the framework provides a mechanism for providing the end-user with a level of knowledge about the inner workings of the simulation far beyond what is provided in most numerical simulation tools.

The customers of the VnV framework include the commercial and government entities that develop large-scale numerical models and simulations (e.g., ORNL, Idaho National Laboratory, AFRL, ANSYS, CD-adapco, universities). For these customers, RNET will provide training, support, and integration services to integrate the VnV framework into new and existing code bases, as well as contract based services for extending the toolkit to fit specific needs. For these customers, the benefits of the VnV framework include; explainable numerical simulations that increase trust in the solution (VnV enabled simulations produce both a traditional result and a description of how the solution was obtained and why it can be trusted), runtime V&V configuration (reduces setup time and allows the users of the code to iteratively build a robust V&V regimen without ever touching the source code of the simulation), a robust collection of V&V tools (improves efficiency by providing statistical assertions optimized for performance in a large scale distributed settings), automated production grade documentation generation (reduces time to market and reduces the burden of meeting the V&V reporting requirements), and cross-library support (multilingual, cross library V&V testing provides a simple mechanism for facilitating end-user V&V at all levels of the simulation hierarchy). While focusing on end-user V&V, the features provided by the VnV framework benefit both simulation developers (our customers) and end-users (users of the simulation software). Additionally, including end-user V&V features into their products will provide a differentiating feature to their end-users.

The true beneficiaries of the V&V framework are the end users of the advanced numerical simulation products. By reducing the burden associated with end-user V&V , the VnV framework will afford these researchers and engineers with the knowledge required to drive the next generation of technological advancements.

1.3 Phase I and Feasibility Demonstration

The goal of the Phase I effort was to demonstrate the feasibility of a framework that facilitates end-user solution verification in advanced numerical simulation. To that end, the project team spent the majority of the Phase I project developing a prototype of the VnV toolkit.

1.3.1 The VnV Injection Point System

The injection point system was written in C++ with a focus on minimizing the amount of code needed to insert injection points into existing code bases.

Integrating of an injection point into an existing code is a three step process; (1) include the header file, (2) place injection point in the code and (3) write the output template. Figure 2 shows a function that has been augmented with a single stage injection point. Developers insert injection points using the INJECTION_POINT macro. The format for this macro is:

```
INJECTION_POINT( <name>, <stage>, <type> <variable>, ...)
```

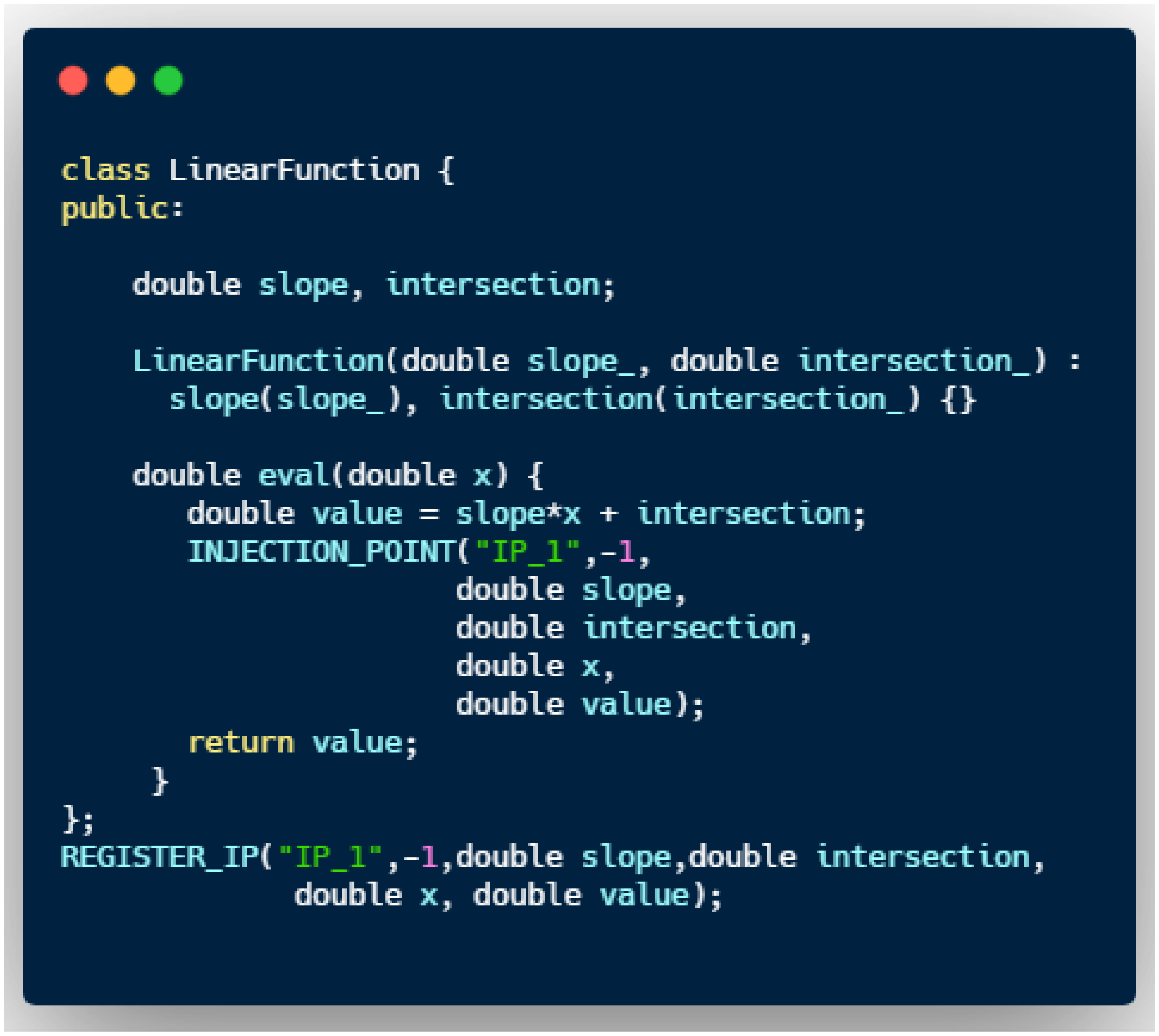
Here, "name" is the id that will be used to define the injection point in the configuration files and the final reports. This name must be unique across all injection points in an executable. The stage parameter is an integer value that defines the step that this injection point belongs to. Using this parameter, the developer can set up multi-staged injection point testing. This allows for data collection across multiple code locations.

The developer defines the variables available for inspection using the variable parameter. During preprocessing, the macro expands this variable definition as

```
 <type> <variable> --> "type", (void*) &variable
```

At runtime, the VnV injection point factory compares the "type" of each variable against the type of variables supported by each VnV test. When a match is found, the framework casts the void* pointer back to its correct type and forwards it to the test for processing. There are several risks associated with this approach, the most important of which is the memory corruption the could occur from a bad cast. Addressing these issues is a major goal of the the Phase II project (see Section 2.2).

The final step in the creation of an injection point is to write the output template. This template is used to populate the final VnV report. The output template is a YAML file containing a markdown formatted description of the injection point. There is no requirement that a output template be supplied, however, the more information that can be provided, the more informative the final VnV report will be. The VnV framework supports a custom markdown format that includes a number of data visualization extensions (2D charts, 3D visualization, Tables, latex, etc). Most importantly, the custom markdown format provides a mechanism for automated post-processing of the data collected during VnV testing.



```
class LinearFunction {
public:

    double slope, intersection;

    LinearFunction(double slope_, double intersection_) :
      slope(slope_), intersection(intersection_) {}

    double eval(double x) {
       double value = slope*x + intersection;
       INJECTION_POINT("IP_1",-1,
                      double slope,
                      double intersection,
                      double x,
                      double value);
       return value;
     }
};
REGISTER_IP("IP_1",-1,double slope,double intersection,
             double x, double value);
```

Figure 2: A code Snippet showing a member function enhanced with a single stage injection point called "IP_1". This injection point is declared inside a member function designed to evaluate a linear function at a particular value of x.

1.3.2 The VnV Testing Interface

The second facet of the injection point system are the V&V tests. The test interface was developed under the assumption that tests should be loaded at runtime and defined independently of the source code. As such, the test interface was built using a C++ plugin pattern.

The framework includes a library generation script that will automatically build the directory structure and makefiles required to build a testing library. Once the library has been initialized, the user can begin to develop individual tests.

To assist in the development of tests, and to avoid issues with incorrect type-casting, a test generation script is also included. This script automatically generates the boiler plate code required to implement the testing interface while also taking care of the required typecasting. With this, implementing a custom test requires the developer to:

- **Implement the "declareParameters" function**. This function defines the parameters required to perform the test. For example, the test in Figure 3 requires two doubles; one called slope and the other called intersect. The user will map the injection point variables to the test variables using the test configuration file.
- **Implement the "runTests" function**. The variables passed to the test are direct pointers to the variables in the code. Hence, tests should not modify the pointers in any way; however, the Phase I prototype does not strictly enforce this requirement. Beyond that requirement, there is not limit on the procedures that can be run inside a test. All data required for post-processing should be output inside this function using the supplied ADIOS engine.
- **Write the output template.** Like injection points, all tests should be accompanied by a output template. The output can be used to automatically post-process and visualize the data collected during each test.

1.3.3 The Runtime Module

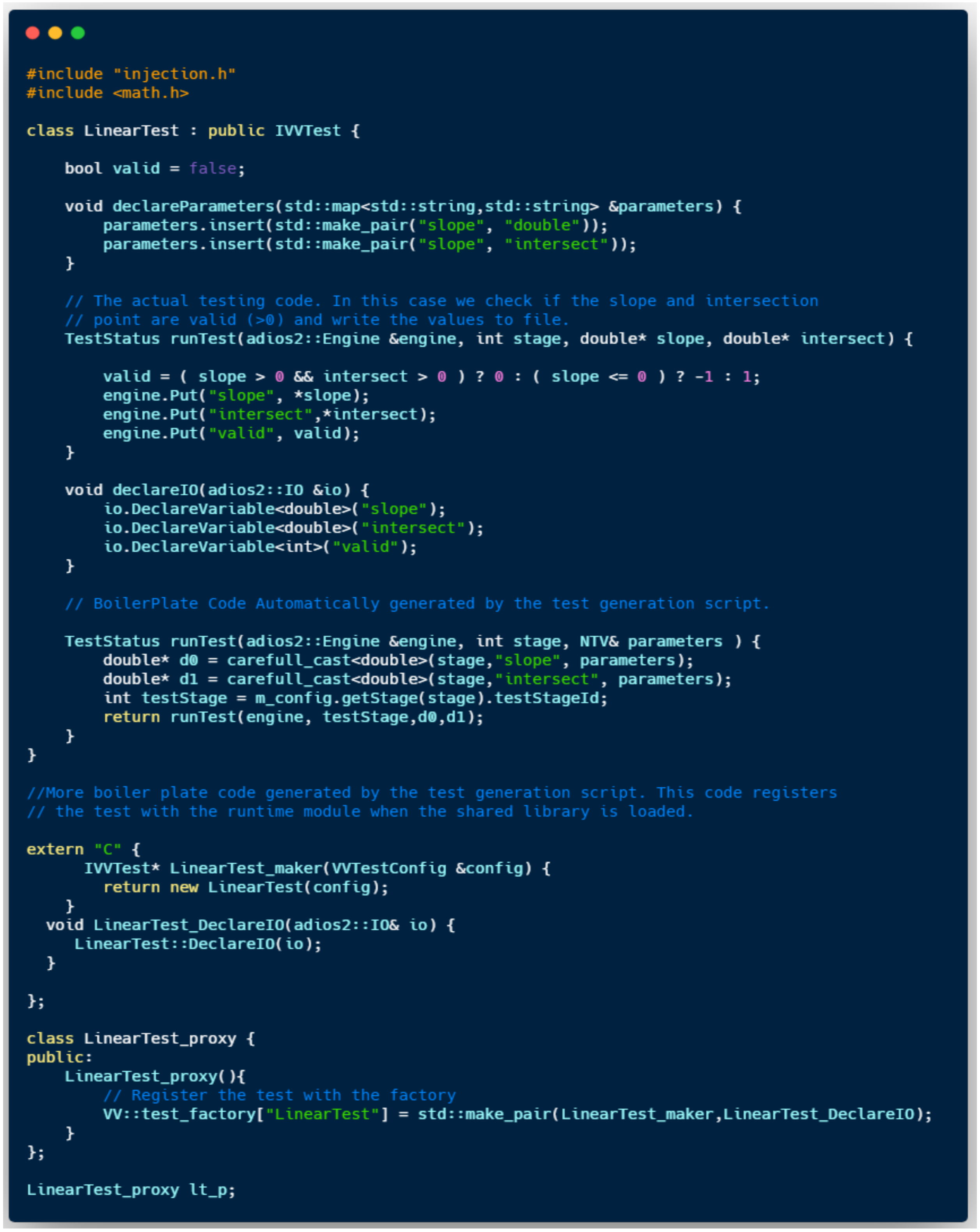
Once the injection points have been declared and the tests created, the next step is to develop the XML test configuration file. Using this XML file, users can configure which testing libraries to load and which tests to run. The test configuration file is used to map injection point parameters to test parameters. For example, Figure 4 shows a portion of a test configuration file that inserts the LinearTest function shown in Figure 2 into the "IP_1" injection point defined in Figure 3. Please see the final report for a full description of the XML format supported by the Phase I prototype.

Turning on VnV testing across all libraries linked to the executable is as simple as calling an initialization function with a valid input configuration file.

1.3.4 Automated Report Generation

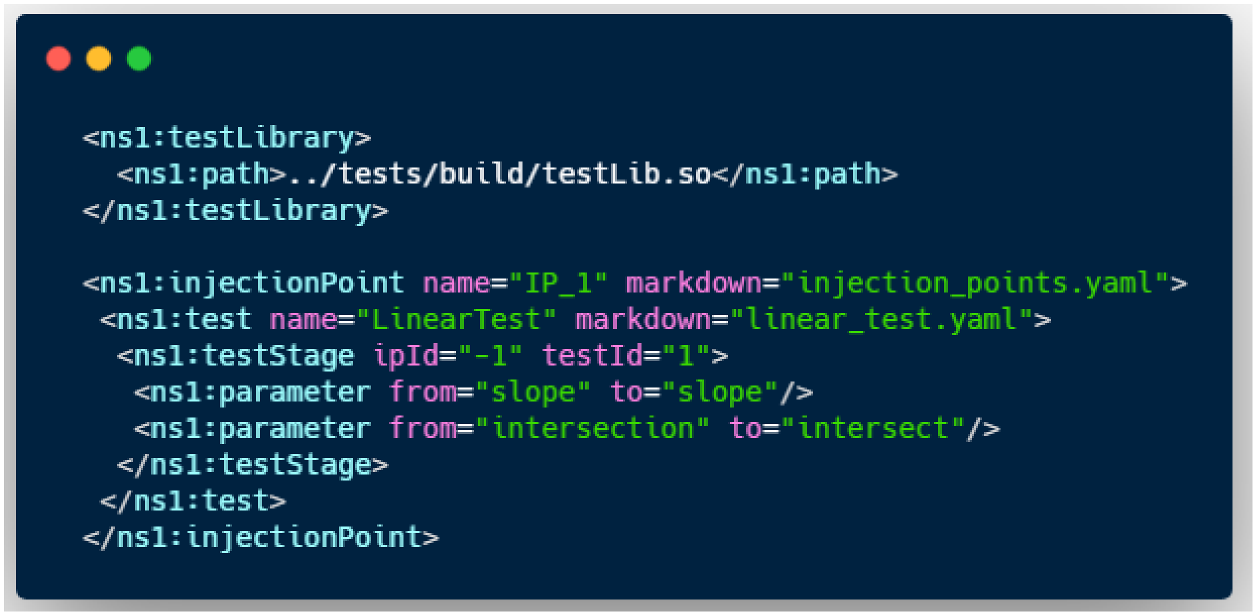
The final step in the VnV pipeline is the automatic report generation. At the core of the report generation system is a custom markdown extension. This extension supports a range of custom data visualization features that interact directly with the output data in the ADIOS2 output file. For example, users can specify markdown that renders as an interactive 2D visualization automatically populated with data obtained during VnV testing (the line chart shown in Figure 5 is generated using this functionality). Other features include 3D visualization using VTK.js and search-able tables with tabular.js. The extension also includes support for running custom python scripts during compilation. This allows for infinite possibilities with regard to processing the test outputs. For example, a user can write markdown that, during compilation, runs a paraview script and then display the results using the VTK.js.

Figure 5 shows a screen-shot of a VnV report generated using this approach. The main layout consists of two components; the index and the content. The index in generated directly from the VnV output file. Each entry in the index represents an injection point that was reached during the execution of the simulation. The content section is generated automatically from the YAML specification files. In this case, the



```
#include "injection.h"
#include <math.h>

class LinearTest : public IVVTest {

    bool valid = false;

    void declareParameters(std::map<std::string,std::string> &parameters) {
        parameters.insert(std::make_pair("slope", "double"));
        parameters.insert(std::make_pair("slope", "intersect"));
    }

    // The actual testing code. In this case we check if the slope and intersection
    // point are valid (>0) and write the values to file.
    TestStatus runTest(adios2::Engine &engine, int stage, double* slope, double* intersect) {

        valid = ( slope > 0 && intersect > 0 ) ? 0 : ( slope <= 0 ) ? -1 : 1;
        engine.Put("slope", *slope);
        engine.Put("intersect",*intersect);
        engine.Put("valid", valid);
    }

    void declareIO(adios2::IO &io) {
        io.DeclareVariable<double>("slope");
        io.DeclareVariable<double>("intersect");
        io.DeclareVariable<int>("valid");
    }

    // BoilerPlate Code Automatically generated by the test generation script.

    TestStatus runTest(adios2::Engine &engine, int stage, NTV& parameters ) {
        double* d0 = carefull_cast<double>(stage,"slope", parameters);
        double* d1 = carefull_cast<double>(stage,"intersect", parameters);
        int testStage = m_config.getStage(stage).testStageId;
        return runTest(engine, testStage,d0,d1);
    }
}

//More boiler plate code generated by the test generation script. This code registers
// the test with the runtime module when the shared library is loaded.

extern "C" {
      IVVTest* LinearTest_maker(VVTestConfig &config) {
        return new LinearTest(config);
    }
  void LinearTest_DeclareIO(adios2::IO& io) {
     LinearTest::DeclareIO(io);
  }

};

class LinearTest_proxy {
public:
    LinearTest_proxy(){
        // Register the test with the factory
        VV::test_factory["LinearTest"] = std::make_pair(LinearTest_maker,LinearTest_DeclareIO);
    }
};

LinearTest_proxy lt_p;
```

Figure 3: An example of a custom VnV test. In this case we implement a test that checks that the slope and intersection point of the linear function are positive and writes the result to file.



```
<ns1:testLibrary>
  <ns1:path>../tests/build/testLib.so</ns1:path>
</ns1:testLibrary>

<ns1:injectionPoint name="IP_1" markdown="injection_points.yaml">
 <ns1:test name="LinearTest" markdown="linear_test.yaml">
  <ns1:testStage ipId="-1" testId="1">
   <ns1:parameter from="slope" to="slope"/>
   <ns1:parameter from="intersection" to="intersect"/>
  </ns1:testStage>
 </ns1:test>
</ns1:injectionPoint>
```

Figure 4: An example of the XML configuration file for attaching test to injection points at runtime. In this case, we attach the LinearTest test shown in Figure 3 to the ”IP_1” injection point shown in Figure 2.

user has completed a test titled “Linear Function Plotter” inside an injection point called “Linear Function Constructor”.

The key point to note here is that this interface was generated automatically from the VnV output file.

In summary, during Phase I, the project team created a functioning prototype of the VnV framework that provides:

- A clean mechanism for inserting injection points in existing codes
- A simple interface for defining custom tests
- A report generation system that automatically creates an interactive server-less web report based on the VnV output.

As will be described in the work-plan, the goal of the Phase II project will be to take this initial prototype and extend, harden and optimize it for use in high performance computing applications.

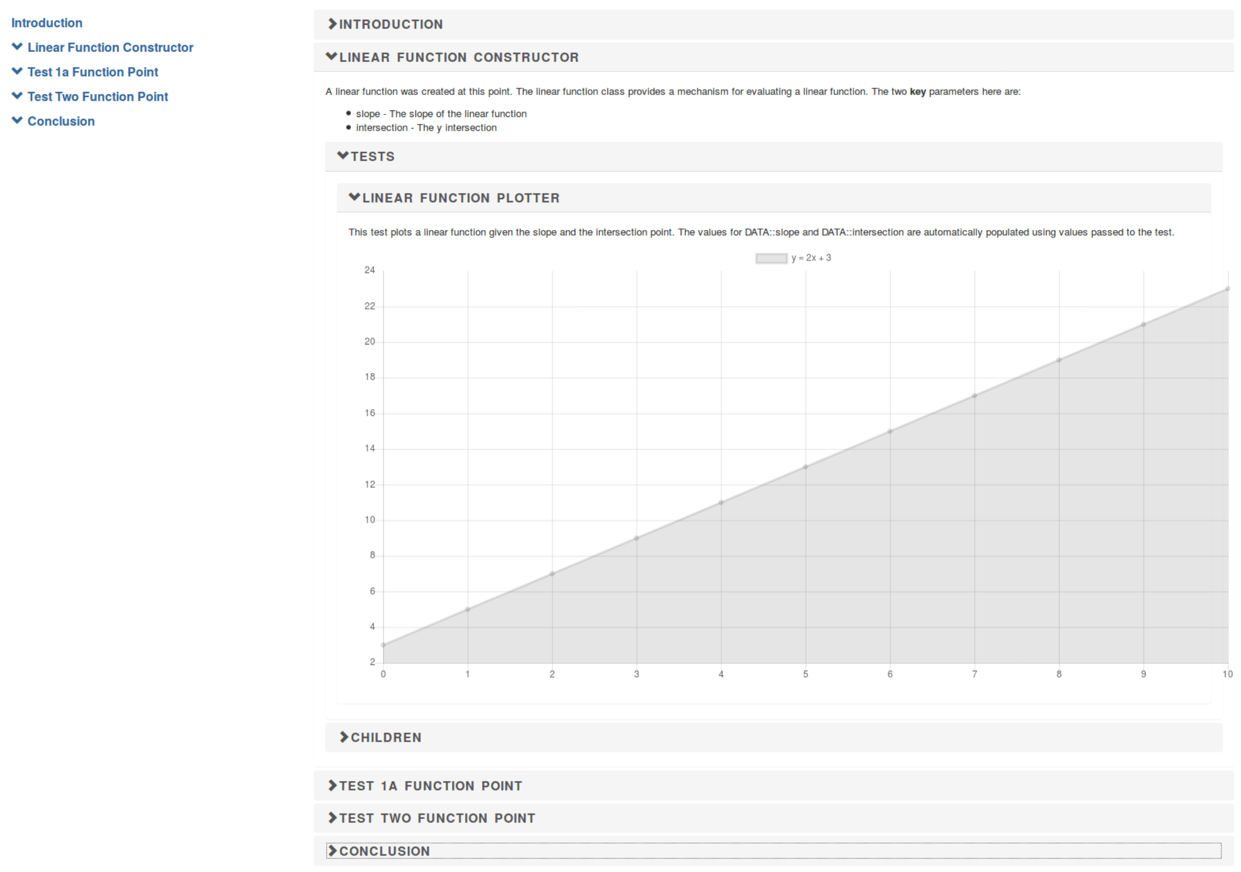
2 The Phase II Project

2.1 Technical Objectives

The requirements being addressed include the development of a robust framework for in-situ verification and validation in general purpose numerical simulation packages. In particular, the objective of this project is to address the need for tools that automate verification of end-user numerical solutions in the NEAMS toolkit. The Phase II project will meet this need by developing a framework that facilitates the creation of explainable numerical simulations that produce a final solution *and* a detailed report on why that solution should be trusted.

Toward this goal, RNET Technologies and ORNL will pursue the following objectives:

1. {Harden and extend the prototype developed during Phase I. In particular, the Phase II effort will focus on developing more robust mechanisms for specifying injection points inside existing code bases. }
2. {Develop mechanisms for efficient in-situ comparisons between data in a distributed environment. In-situ comparison of variables with experimental and/or analytical results will significantly reduce the amount of IO required in V&V testing, while also providing a fine grained mechanism for detecting at what point a solution diverges from the expected result.}


Introduction
Linear Function Constructor
Test 1a Function Point
Test Two Function Point
Conclusion

INTRODUCTION

LINEAR FUNCTION CONSTRUCTOR

A linear function was created at this point. The linear function class provides a mechanism for evaluating a linear function. The two **key** parameters here are:

- slope - The slope of the linear function
- intersection - The y intersection

TESTS

LINEAR FUNCTION PLOTTER

This test plots a linear function given the slope and the intersection point. The values for DATA::slope and DATA::intersection are automatically populated using values passed to the test.

CHILDREN

TEST 1A FUNCTION POINT

TEST TWO FUNCTION POINT

CONCLUSION

Figure 5: An example of the VnV report generated automatically from the VnV output files and the injection point and test specification files.

3. {Minimize execution times through job based parallelism. This will shift the burden of V&V testing out of the main simulation, allowing for faster runtime, especially for testing methods that require the same function to be executed multiple times.}
4. { Develop a robust set of generic V&V tests. The development of these tests (e.g., mesh refinement studies, the method of manufactured solutions, sensitivity analysis, uncertainty quantification) will equip the users with the tools required to perform end-user V&V .}
5. {Demonstrate the value of the VnV framework as a component in the NEAMS tool-chain (MOOSE, libMesh and PETSc). Integration into these tools will answer the NEAMS call for tools that support end-user verification of numerical simulations. Integration into the NEAMS workbench will provide access to the tools in the seamless manor users of the workbench have come to expect. }

2.2 Work Plan

The goal of the Phase II project will be to develop a fully functional, efficient, battle tested framework for integrating advanced end-user verification and validation into general purpose numerical simulation packages. In what follows, we outline the work required to achieve this goal.

2.2.1 Hardening and Optimization of the Injection Point System

The injection point system developed during Phase I relies on C style Macros and string based pointer casts. In a *trusted* environment, this is an efficient (void* casts are almost free) and portable (it is written in C) approach. However, there are three main weaknesses need to be addressed prior to production:

- **String based pointer casts:** The system developed in Phase I uses developer provided strings to infer the type of each variable. Under the hood, the injection point system uses these strings to ensure compatibility between test parameters and injection point variables. The key issue is that the strings specified at each injection point cannot be verified during compilation. This will causes issues in cases where, say, the developer changes the type of a variable, but forgets to update the type string in the injection point declaration.
- **Restrictive type specifications:** The current system uses C compliant pre-processor macros to simplify the process of describing injection points and variables. The benefit of this approach is that the injection points can be compiled into any application without significant changes to the build system. The downside is that the single-pass, text based pre-processing supported by the C preprocessor places a significant restriction on the functionality that can be imparted.
- **Constant Correctness** The current system provides the testing algorithms with direct access to the data structures and variables specified at each injection point. This access allows for efficient, unrestricted testing of the data structures. There are some exciting use cases for this functionality (parallel debugging, computational steering, etc.); however, from the perspective of V&V, it is imperative that testing routines do not alter the trajectory of the simulation in any way.

In Phase II, the project team will investigate and develop an annotation based system for defining injection points. This annotation system will provide users with full control over what variables can be accessed at each injection point, including the ability to provide access to internal components of data structures, describe the domain decomposition of distributed arrays and to complete pre-test processing. The annotation system will also provide a mechanism for injection point detection in non-object orientated programming languages where runtime injection point detection cannot be completed. However, the primary benefit of this new system will be that it will automatically detect the type of the variables tagged for inspection at each injection point.

To implement this annotation system, the project team will create a set of custom pre-processor directives (i.e., pragmas) using Clang. Clang is a compiler front end for the C family of languages. It is a well supported, well documented compiler package designed as a drop in replacement for GCC. We we utilize Clangs library API for implementing custom Pragma routines to implement a pre-processor that transforms the annotations into valid injection point specifications.

To address the issues with constant correctness, the team will develop support for fine-grained regression testing. This functionality will allow users to track the progress of a simulation against a VnV output file that was generated without in-situ testing. This will allows users to compare the current state of the simulation against an expected result at every level of the simulation.

2.2.2 Efficient Statistical Comparisons in Distributed Settings

One of the major benefits of the VnV framework is that the tests are executed inside the simulations distributed environment. This allows for the development of efficient, parallel testing algorithms. To capitalize on this functionality, the Phase II effort will include an investigation into efficient statistical methods for analyzing, asserting and comparing data stored in distributed arrays.

The first step toward working with distributed arrays is to obtain some information about the global partition of the data. This information is readily available in simulations that use regular data decompositions (i.e., block, cyclic, etc.), but it is not generally known in simulations where the data is distributed irregularly across the processors.

The naive approach to determining the global partition is to explicitly form the partition using global MPI communication. This is a robust approach that will be immediately applicable in a large number of situations. However, the large storage cost required to build the global partition (O(P), where P is the number of processors) is likely to become an issue in peta- and exa-scale environments [5].

To that end, the Phase II effort will include a detailed analysis of the optimal algorithms for forming the global partition in a distributed setting. This will include the implementation and profiling of the aforementioned naive approach, as well as an investigation into more advanced approaches parallel rendezvous algorithms such as forming a hash based distributed directory [9] and the assumed partition algorithm [5].

The second step will be to develop efficient algorithms for comparing and analyzing distributed data based on that partition. This will include the development of the VnV Testing suite for distributed Arrays. This testing suite will include a variety of analysis routines for analyzing data including, means, std deviation, variance, co-variance, etc. A collection of functions implementing matrix based metrics will also be included (e.g., one norms, symmetry, positive definiteness, etc).

A key goal of the Phase II effort will be to develop support for comparing solutions obtained in distributed arrays with data stored on disk. This will be achieved using the ADIOS2 read-write API. In this case, the global partition determined above will be used in conjunction with ADIOS to distribute the correct data to each processor. The project team will also perform a detailed analysis of existing third party data transfer and interpolation tools to determine the best approach to efficiently comparing experimental data with the data in the distributed arrays. The project team will also investigate the feasibility of implementing a multi-cast reduction network to perform reductions and comparisons on the data.

2.2.3 Reducing Run-times with Job Parallelism

Performing V&V tests in a distributed environment will be expensive, both computationally and due to the data movement required to deal with the domain decomposition employed by the application. To address this problem, the VnV toolkit will support the offloading of tests to external processes. This functionality will allow for speedup through job parallelism for expensive testing routines like sensitivity analysis and mesh refinement, while also allowing for specific tests to be run on an optimal architecture (e.g., a test involving image processing could be offloaded to a GPU enabled architecture).

The first step toward supporting test offloading will be to implement an effective mechanism for transferring the required data from the simulation to the external testing processes. To do this, the project team will use the ADIOS 2 Sustainable Staging Transport (SST). The SST is a classic streaming data architecture that allows for direct connection between data produces and data consumers.

The VnV toolkit will use the SST engine to stream the required testing data to a VnV Testing Manager. The test manager will determine the best available resource for running the given test and launch the job. To do this, the test manager will utilize the extensive support for integrating with remote job schedulers available in the Eclipse Parallel Tools Platform (PTP).

The PTP project provides an integrated development environment to support the development of parallel applications written in C, C++, and Fortran. Eclipse PTP provides; support for the MPI, OpenMP and UPC programming models, as well support for a wide range of batch systems and runtime systems, including PBS/Torque, LoadLeveler, GridEngine, Parallel Environment, Open MPI, and MPICH2.

In this case, PTP provides a simple interface for launching tests as separate processes. The open research question to be answered during the Phase II effort is to determine in which situations test offloading is beneficial. In particular, the project team will look to implement heuristic algorithms that detect when the cost of streaming the required testing data outweighs the performance benefits associated with offloading the tests.

2.2.4 Integrate Third Party Tools for Mesh Refinement, UQ and Sensitivity Analysis.

Mesh refinement studies, uncertainty quantification (UQ) and sensitivity analysis (SA) are all essential components of a robust V&V regimen. To that end, the Phase II effort will include the development of a VnV testing library that integrates with third party tools to facilitate testing using these approaches.

UQ and SA will be developed using DAKOTA. DAKOTA provides a set of black-box tools for performing parameter optimization, UQ and SA. The Phase II effort will include the development of a flexible interface for setting up and running DAKOTA tools at injection points, and the development of the output templates for displaying the results. Most high performance finite element packages support some level of adaptive mesh refinement (e.g., libMesh, Fenics, MFEM, etc.), hence, rather than trying to patch in third party tools, the mesh-refinement tool will be developed as a generic interface for interacting with existing mesh refinement functionality.

2.2.5 Integration with NEAMS tools and the NEAMS workbench

To demonstrate the value of the VnV framework, the project team will integrate the VnV framework into the open-source components of the NEAMS tool-chain (PETSc, libMesh and MOOSE). Doing so will provide ample opportunities for testing, while also allowing us to demonstrate the performance of the toolkit in real codes with real applications. Moreover, this integration will provide a road map for integration into MOOSE based NEAMS applications like BISON.

Integration into NEAMS tool-chain will be a three step process:

- The first step will be to insert injection points into key locations in the MOOSE, libMesh and PETSc source codes. The optimal locations for inserting these injection points will be determined after detailed profiling of example codes; however, some options include during each linear solver iteration, during each nonlinear iteration, during finite element matrix construction (if it exists) and inside the function for calculating the action of the Jacobian on a vector (for matrix free problems).
- The second step will be to allow users to configure the VnV testing directly in the MOOSE input file using the MOOSE input file syntax. This will provide users of MOOSE with a seamless mechanism for setting up and running tests. This will involve writing a custom MOOSE action for setting up and configuring the VnV runtime module.
- The third step will be to enable context-aware auto-complete for MOOSE based VnV configuration files in the NEAMS workbench. The workbench has integrated support for extracting the information required to setup input file verification and auto-complete from MOOSE applications; however, there will likely be some additional work required to correctly setup this up for tests stored in external libraries. In particular, the current system requires that the tests adhere to a specific XSD specification, but there is not yet a system for extracting the exact parameters required to inject a specific test.
- The last step will be to provide support for viewing the final VnV reports in the NEAMS workbench. The project team demonstrated how the QWebEngineView can be used to display the HTML VnV report in a QT window. The Phase II effort will extend that demonstration by providing the callback function required to interact with the NEAMS workbench interface (menus, docks, etc).

2.3 Performance Schedule and Task Plan

RNET would like to present the project ideas and research plan to the DOE Program Manager and other interested scientists. The meeting will be used to discuss features and to identify the specific NEAMS applications and computer resources that will benefit from this project. This meeting will be scheduled soon after the Phase II contract is awarded. The meeting can be hosted at RNET, a DOE site suggested by the Program Manager or via a teleconference.

RNET will submit all reports as required by the contract (e.g., annual reports, a continuation report, summary reports, and a final report) to the DOE program manager and other interested DOE scientists.

The research and development topics described in Section 2.2 will be addressed by the tasks described in the remainder of this section. Most tasks require active collaboration between RNET and ORNL. Figure 6 summarizes at a high level the dependencies among tasks and approximate anticipated task durations. The duration of the Phase II project is 104 weeks. Specific details are included in the description of each task.

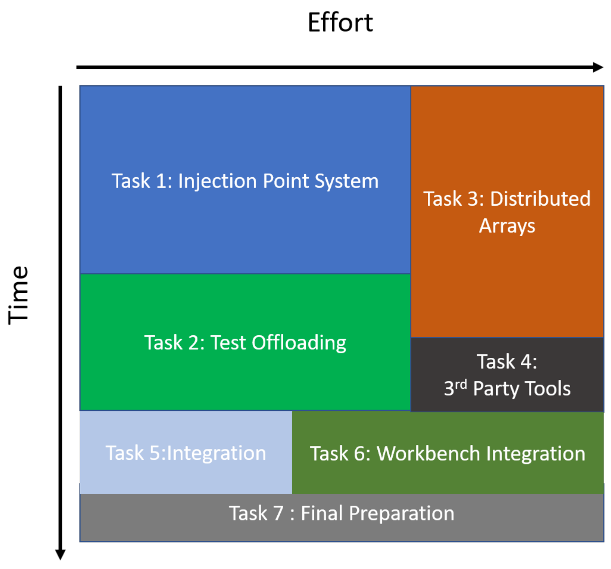

Effort
Time
Task 1: Injection Point System
Task 3: Distributed Arrays
Task 2: Test Offloading
Task 4: 3rd Party Tools
Task 5:Integration
Task 6: Workbench Integration
Task 7 : Final Preparation

Figure 6: Overview of task dependencies and timeline.

2.3.1 Task 1: Development of the Annotation based Injection Point System

{In this task, the project team will develop the Annotation system for specifying injection points and describing injection variables. This will include the development of the custom LLVM compiler extensions required to process these annotations. At the end of this task, users will be able to specify injection points and injection point variables using either the new Annotation point system, or using the more portable, but less robust Phase I approach.

To test the Annotation based injection point system, the project team will write injection points at several key locations in the MOOSE software stack, including inside the source code of several MOOSE modules, the core MOOSE framework, libMesh and PETSc. This will allow us to demonstrate the cross-library potential of the Annotation based system, while also acting as the first step toward integration of the tools a number of the NEAMS tools.}

{ RNET will work on the implementation for this task and ORNL will provide inputs and guidance. }

2.3.2 Task 2: Develop methods for offloading tests to external processes.

{In this task, the project team will add support for test offloading using the ADIOS2 SST architecture. This will include the development of the VnV test manager and integration with the PTP platform, a task which will likely involve custom modifications to the PTP platform. Once implemented, test offloading will be tested using the VnV enabled MOOSE software stack developed in Task 1. The goal of this task will be to see a significant speedup when using test offloading over the standard in-situ testing routines. }

{ RNET will work on the implementation for this task with ORNL providing assistance with ADIOS and any modifications to PTP. }

2.3.3 Task 3: Development of efficient statistical V&V tools for distributed arrays.

{In this task, the project team will investigate the optimal approaches for conveying information about the data decomposition. This will involve robust analysis of three approaches to determining the global partition; explicitly forming the global partition; building a distributed directory; and using an assumed partition algorithm. This task will also include the development of a number of efficient statistical tools for asserting the state of the data in distributed arrays.

At the completion of this task, users will be able to efficiently compare and analyze distributed arrays using the VnV framework. To test these implementations, the project team will set up tests for performing assertions on various PETSc vectors and matrices. This testing will compare the performance of each of the global partition methods and each of the analysis tools.}

{ RNET and ORNL will collaborate on this task. }

2.3.4 Task 4: Development of Generic tools for Mesh refinement, Uncertainty quantification and Sensitivity Analysis.

{In this task, RNET will implement generic V&V tools for mesh refinement, uncertainty quantification and sensitivity analysis. In the case of mesh refinement, the approach taken will be to create a generic interface for interacting with the automatic mesh refinement functionality that already exists in finite element libraries like LibMesh and Fenics. The overall goal is to create a generic VnV test that can be attached to the main function of the executable such that it automates the process of running mesh refinement and mesh convergence studies. In the case of UQ and SA, the project team will develop an interface for specifying the tools available in DAKOTA as VnV tests. }

{ RNET will be responsible for this task with ORNL providing input and guidance. }

2.3.5 Task 5: Integration into the NEAMS tool-chain

{ In this task, the program team will integrate the VnV framework into a variety of real applications. Initially, this testing will be completed in tools used heavily in the NEAMS toolkit; MOOSE, libMesh and PETSc, but other third party applications will also be investigated. To ensure seamless integration with the MOOSE tools, the project team will re-implement the XML based configuration file using the MOOSE input file format. This will allow for the configuration of the VnV tests in a MOOSE application directly from the input file.

The goal of this task will be to generate informative, production quality VnV reports for a number of examples available in the MOOSE testing suite. Doing so allows us to test every facet of the proposed framework, while also acting as the first demonstration of the value provided by the framework. These results of these tests will be hosted on the RNET website as they become available. }

{ RNET will be responsible for this task and ORNL will provide guidance on various technical implementations and details. }

2.3.6 Task 6: Development of an interface for the NEAMS workbench

{ In this task, the project team will integrate the toolkit directly into the NEAMS workbench. This will be a two stage process. First, the project team will implement the required interface files for enabling the context aware auto-complete features available in the NEAMS workbench. MOOSE based input files are already largely supported in the workbench; however, there will likely be some issues with determining which tests are applicable at which injection points. Second, we will create the interfaces for viewing the HTML based VnV reports inside the QT based workbench. As demonstrated in Phase I, this will be completed using the QWebEngineView components support for displaying HTML files inside QT windows. }

{ RNET will be responsible for this task }

2.3.7 Task 7: Preparation for the First Release

{In this task, the project team will prepare the package for its first release. This will include writing documentation ,user manuals and tutorials.

An important goal of this task will be to develop a mechanism for reducing the size of the VnV reports. In particular, this task will involve modifying the report generation system such that it maximizes data reuse in situations where an injection point or test is encountered multiple times within a simulation. Support for lazy-loading of content in collapsed sections will also be implemented. Such features are not critical to the functionality of the toolkit; however, they will significantly improve the usability of overall product. The custom markdown format will also be hardened, extended and released as part of this task. }

{ RNET will work on the implementation for this task }

2.4 Facilities/Equipment

2.4.1 RNET Facilities

RNET has the necessary office equipment to manage an SBIR/STTR contract including networks, workstations, and accounting software. In addition, RNET has the tools (software and hardware) to evaluate and develop the technologies proposed here.

RNET currently has 9 development computers and a 10-node development cluster that can be used for development and testing in this effort. Each cluster node has two quad-core or hexa-core XEON CPUs, 24-32GB of DRAM, 500+GB of local disk. Two data networks are available, a COTS 1 Gbps Ethernet network and a 10 Gbps Ethernet network. The Linux development nodes and the RNET cluster have the necessary Linux/GNU toolchains and development environments including; GNU tool chain, Microsoft .Net Framework, and Java Standard Edition.

2.4.2 ORNL Facilities

The Oak Ridge National Laboratory (ORNL) hosts three petascale computing facilities: the Oak Ridge Leadership Computing Facility (OLCF), managed for DOE; the National Institute for Computational Sciences (NICS) computing facility operated for the National Science Foundation (NSF); and the National Climate-Computing Research Center (NCRC), formed as collaboration between ORNL and the National Oceanographic and Atmospheric Administration (NOAA) to explore a variety of research topics in climate sciences. Each of these facilities has a professional, experienced operational and engineering staff comprising groups in high-performance computing (HPC) operations, technology integration, user services, scientific computing, and application performance tools.

ORNL also has the Compute and Data Environment for Science (CADES) which is a fully integrated infrastructure offering compute and data services for researchers lab-wide. We will work with appropriate program managers to apply for allocation requests as appropriate.

The ORNL computer facility staff provides continuous operation of the centers and immediate problem resolution. On evenings and weekends, operators provide first-line problem resolution for users with additional user support and system administrators on-call for more difficult problems. ORNL also has state-of-the-art visualization facilities that can be used on site or accessed remotely.

3 Consultants and Subcontractors

Oak Ridge National Laboratory(ORNL) is the Research Institution for this proposal and will serve as a subcontractor for this SBIR.

3.1 Gregory Watson

Gregory Watson, PhD, is a Senior Research Scientist in the Computer Science Research Group at Oak Ridge National Laboratory. Dr. Watson's research interests include programming tools and development environments for high performance and scientific computing, software engineering practices, reproducibility, and education and training for scientists. Dr. Watson is the founder of the Eclipse Parallel Tools Platform, a project that was originally started as a collaboration between Los Alamos National Laboratory and IBM in 2004, and that continues to be used across laboratories, academia, and industry. He is also a founding member of the Eclipse Science Working Group, and project leader of the Eclipse Science Top Level Project. Dr. Watson has considerable experience developing and implementing efficient parallel debugging software for high performance computing environments and has a wealth of experience in the development of highly scalable tools and communication frameworks for peta-scale high performance computing systems.

4 Principal Investigator and other Key Personnel

4.1 Ben O'Neill, PI

Ben O'Neill is a Research Scientist at RNET and will be the PI on this project. Ben is a full time employee at RNET and has sufficient time to dedicate to this project. Ben is a permanent resident in the USA (citizen of New Zealand). The proposed work does not include any Export Control restriction, as such, this work visa should be sufficient. In addition to the current project, Ben is the lead developer in RNET's Cloudbench project which aims to develop a web-enabled interface for remote execution and visualization for nuclear physics tools. Ben was also heavily involved in the development of the SolverSelector framework for facilitating automatic linear solver selection in high performance applications through machine learning. His background is in Applied mathematics with a focus on high performance computing and parallel-time integration. His thesis work involved the optimization and implementation of a parallel time integration codes for nonlinear PDEs including implementing a fully adaptive and parallel space-time solver using the Fenics finite element package.

4.2 Gerald Sabin

Dr. Gerald Sabin, Project Manager at RNET, will be the senior advisor for this project. Dr. Sabin is a full time employee of RNET, and has sufficient time to dedicate to project tasks as indicated in the cost proposal. Since he is a US Citizen, he can undertake relevant integration work in Export-Controlled areas of the project, if necessary. Currently, he is working on several Scientific Computing (HPC) SBIR/STTR projects at RNET. He is the PI for this Phase II SBIR Cloudbench project (DE-SC0015748) and the ongoing Phase II DOE SBIR for the Automated Solver Selection for Nuclear Engineering Simulations. He has also worked on distributed memory, GPU, multi-core and SIMD optimizations to the Air Force's Kestrel code (DOD Contract#:FA9550-12-C-0028) and has also been involved in developing fine-grained power profiling hardware and software tools for HPC application profiling (DOE Contract#:DE-SC0004510). He has also been the PI on several other related projects including a NASA Phase I project developing SIMD optimizations for Monte Carlo codes (NNX14CA44P), developing parallelization optimizations for PETSc (DOE Contract#: DE-SC0002434), developing data virtualization support and bitmap indexing for massive Climate Modeling data sets (DOE Contract #:DE-SC0009520), and developing the SmartNIC software stack for application-aware network offloading (DOE Contract#: DE-FG02-08ER86360).

5 Related Work

RNET and ORNL have past and current experience in several SBIR/STTR projects on modeling and simulation, high performance computing, and large data formats. Some of these projects are briefly described below.

5.1 RNET's Related Work

5.1.1 Automated Solver Selection for NEAMS Tools

RNET in collaboration with University of Oregon (Prof. Boyana Norris) is developing an add-in feature for the NEAMS toolkit being developed by the Department of Energy. This work is being done as part of DOE Phase II STTR project (Contract Number DE-SC0013869). This add-in feature being developed by RNET leverages machine learning techniques to automatically select the optimal solver based on run-time dependent features of the problem and the underlying compute architecture with minimal runtime overhead in solver selection during the course of NEAMS simulations.

5.1.2 Cloud-based Scientific Workbench for Nuclear Reactor Simulation Life Cycle Management

The predictive modeling approaches and softwares being continually developed and updated by the DOE nuclear engineering scientists (under programs such as NEAMS, CASL, RISMC etc.) need to be efficiently transferred to the nuclear science and engineering community. An advanced workflow management workbench is required to allow efficient usage from small and large business and research groups. The workbench must manage inputs decks, simulation execution (on a local machine, a High Performance Compute cluster, or a Cloud cluster), intermediate results, final results and visualizations, and provenance of the tools and settings. CloudBench is a hosted simulation environment for large scale numeric simulations. CloudBench will augment existing simulation, Integrated Development Environment, and workbench tools being developed by the DOE and industry. It offers a complete set of simulation management features not available in open tools: sharing of configurations, simulation output, and provenance on a per simulation or per project basis; multi-simulation provenance history to allow simulations to be reconstructed, verified, or extended; and remote access to simulation tools installed on Cloud and HPC resources. The portal enables easy adoption of government codes.

5.1.3 Scaling the PETSc Numerical Library to Petascale Architectures

RNET has developed an extended version of the numerical library PETSc [8] in collaboration with Ohio State University and Argonne National Lab. PETSc is an MPI-based numerical library of linear and nonlinear solvers that is widely used in a variety of scientific domains. With the emergence of multicore processors and heterogeneous accelerators as the building blocks of parallel systems, it is essential to restructure the PETSc code to effectively exploit multi-level parallelism. Changes to the underlying PETSc data structures are required to leverage the multicore nodes and GPGPUs being added to the "cluster architectures".

This project was funded by Department of Energy under the STTR program from August 2010 (Contract Number DE-SC0002434) to May 2013. Dr. P. Sadayappan (OSU) and Dr. Boyana Norris (ANL) have played a key role in this effort by serving as technical advisors. As part of the project, the team has investigated ways for the PETSc library to fully utilize the computing power of future Petascale computers. Novel sparse matrix types, vector types, and preconditioning techniques that are conducive for GPU processing and SIMD parallelization have been integrated into the PETSc library. The matrix vector operations have been optimized for specific architectures and GPUs by utilizing the autotuning tools.

5.2 ORNL's Related Work

Dr. Watson is the main developer for the Eclipse Parallel Tools Platform (PTP). The aim of the PTP project is to produce an open-source industry-strength platform that provides a highly integrated environment specifically designed for parallel application development. The platform provides a standard, portable parallel IDE that supports a wide range of parallel architectures and runtime systems; a scalable parallel debugger; support for the integration of a wide range of parallel tools; and an environment that simplifies the end-user interaction with parallel systems.

In addition, Dr. Watson has played a significant role in the development of GAURD, a parallel relative debugger for high performance systems. Unlike other conventional parallel debuggers ,a relative debugger provides the ability to dynamically compare data between two executing programs regardless of their location and configuration. In GAURD, data comparisons can be performed either using an imperative scheme or a declarative scheme. Imperative comparisons can be performed explicitly by the user when two programs under the control of the debugger are stopped at breakpoints.

A solid foundation has been laid to achieve the objectives of the proposed project. The final piece of the puzzle is the optimization and hardening of these core approaches for efficient execution in real applications. Once this is accomplished, it will be possible to seamlessly integrate functionality for end-user V&V into any general purpose numerical simulation software. The collaboration between ORNL and RNET is a perfect fit to facilitate this final step. RNET brings to the table the required computer science expertise to satisfy the needs of this project, as is evident from a description of their related work, and ORNL brings a wealth of experience developing novel HPC software solutions, developing robust debugging tools for use in HPC environments and working with a range of real application codes.

References

[1] DAKOTA - Explore and Predict with Confidence . https://dakota.sandia.gov/.

[2] DoD VV&A Recommended Practices Guide . https://vva.msco.mil/.

[3] NEAMS Software Verification and Validation Plan Requirements V0. https://preview.tinyurl.com/y8q7zr2v.

[4] Neams: The nuclear energy advanced modeling and simulation program. http://people.nas.nasa.gov/~pulliam/Overflow/Overflow_Manuals.html.

[5] Allison H Baker, Robert D Falgout, and Ulrike Meier Yang. An assumed partition algorithm for determining processor inter-communication. *Parallel Computing*, 32(5-6):394–414, 2006.

[6] MR Collins, Franck J Vecchio, Robert G Selby, and Pawan R Gupta. The failure of an offshore platform. *CONCRETE INTERNATIONAL-DETROIT-*, 19:28–36, 1997.

[7] Bernt Jakobsen and Finn Rosendahl. The sleipner platform accident. *Structural Engineering International*, 4(3):190–193, 1994.

[8] D. Lowell, J. Holewinski J. Godwin, D. Karthik, C. Choudary, A. Mametjanov, B. Norris, G. Sabin, P. Sadayappan, and J. Sarich. Stencil-aware gpu optimization of iterative solvers. *SIAM Journal on Scientific Computing*, 35(5), 2013.

[9] A Pinar and B Hendrickson. Communication support for adaptive communication. *In Proceedings of the 10th SIAM Conference on Parallel Processing*, 2001.