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[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

"Pages TODO and TODO of this document may contain trade secrets or commercial or financial information that is privileged or confidential and is exempt from public disclosure. Such information shall be used or disclosed only for evaluation purposes or in accordance with a financial assistance or loan agreement between the submitter and the Government. The Government may use or disclose any information that is not appropriately marked or otherwise restricted, regardless of source. Proprietary information is marked with blue curly brackets (i.e., { ... }).")

Contents

1 Significance, Background, and Technical Approach

RNET Technologies Inc. (RNET) in Dayton, OH and Oak Ridge National Laboratory (ORNL) are responding to 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 in-situ solution verification and validation in high performance computing applications. In this proposal we will highlight the need for, and the tremendous value of, the proposed tool in high performance numerical simulations like those in the NEAMS toolkit. The goal of the proposed Phase II project will be to develop and demonstrate a production quality implementation of the VnV framework that can deliver this value to a broad spectrum of numerical simulation applications.

1.1 Identification and Significance of the Innovation

The fundamental concept of the VnV framework will be to facilitate the development of *explainable* numerical simulations. That is, the VnV framework will equip developers with all the tools required to create simulations that not only provide a solution, but also a detailed description of how the solution was obtained and why it should be trusted. RNET has extensive SBIR experience in various aspects of High Performance Computing, including performance optimization of numerical softwares and libraries, development of fine-grained power monitoring tools for HPC infrastructure, and the development of software usability tools that enhance the user experience when working with HPC simulations. Dr. Watson, our collaborator at ORNL, is an experienced software professional with extensive knowledge of software design, architecture, and engineering practices. He has significant experience developing parallel debugging software for HPC environments, and is the project leader of the open-source Eclipse Parallel Tools Platform project. 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 fully develop and commercialize the VnV framework.

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 of M&S in the R&D lifecycles of products ranging from $10 polycarbonate toys, up to solar panels, airplane wings and nuclear reactors. With access to large-scale computational resources at an all time high, and with exascale computing resources on the horizon; the role of M&S in the design 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 [**?**], 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, 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 general concensus of these industry specific V&V reports is that V&V should be an iterative, integrated component of the software development lifecycle. 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 realistically account for each and every possibility. This raises significant issues in the development of general purpose numerical simulation packages because, while it may be the responsibility of the program developers of such packages to ensure that their product is mathematically correct and is free of programming errors (so-called bugs), it is ultimately the responsibility of the end user to ensure the solution obtained is an 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 on the products creator, and any attempt to shift the blame to the developers of simulation library X will certainly fall of deaf ears.

End-user V&V is of particular importance 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 [**?**]. An important objective of the NEAMS program is to enable widespread use among the industry, academia, and regulatory communities[**?**]. This objective led to the development of the NEAMS workbench, which has significantly increased the overall user experience of the NEAMS tools. The NEAMS group has also placed a significant emphasis on V&V in the NEAMS toolkit, as outlined in the NEAMS V&V plan (version 0) [**?**]. The result of this is that the NEAMS tools have integrated functionality for input validation (through the workbench and MOOSE), for performing mesh refinement and method of manufactured solution analysis (through MOOSE) and for completing uncertainty quantification and sensitivity analysis with DAKOTA (through the workbench). Despite this, the complex nature of the codes, combined with the expert knowledge required to set up V&V testing and the seemingly infinite array of input parameters (PETSc, the linear solver package used in MOOSE has hundreds of configuration options alone) makes setting up robust end-user V&V an complicated task for all but the most expert users.

To address the difficulties with end user solution verification in the NEAMS toolkit, and the numerical simulation community as a whole, RNET Technologies Inc. (RNET) and Oak Ridge National Laboratory (ORNL) are proposing the development of the VnV toolkit; 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). To do this, the framework will promote the development of *explainable* numerical simulations that, in addition to producing a simulation result, produce a detailed report outlining how the solution was obtained and why it should be trusted.

1.1.2 Product Overview and Technical Approach

In this proposal, we make the distinction between V&V *of* a simulation package and the end-user V&V of a solution obtained *using* a simulation package. These two processes are not independent; indeed, 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 in solutions obtained by end-users. The focus of this proposal and the phase II project will be end-user validation; however, there is no reason that the functionality imparted by the proposed framework could not be used in the V&V process of the overall simulation package as well.

They key issues associated with end-user verification and validation, and the approaches that the VnV framework will take to address them are:

- **In-situ Testing And Analysis:** Unit tests are an extremely effective mechanism for ensuring a algorithm behaves as is expected. However, such testing is an unavoidably discrete process that, by definition, cannot cover every possible outcome. This fact is particularly true for numerical algorithms, where even small changes (e.g., input parameters, mesh geometry. etc.) can cause algorithms to behave unexpectedly (i.e., diverge, converge to the wrong solution, etc.). As such, a robust V&V report should not only include a description of unit tests completed, but also a detailed set of in-situ tests and assertions that run as the simulation proceeds. The VnV framework will include a sophisticated test injection system. At run time, the system will automatically detect, configure and assimilate data obtained from injection points in any VnV equipt library that is linked to the executable. For example, once fully integrated at all levels of the simulation heirarchy, the user of a MOOSE tool would be able to detect, configure, modify and run tests at injection points defined in the source codes of hypre, PETSc, libMesh, MOOSE, etc. This cross library support will allow for in-depth, expert directed, end-user V&V in executables that utilize a range of numerical simulation libraries. A key feature of the framework is that, while the injection points will hard coded into the source code of the individual libraries, the tests themselves are compiled in external shared libraries loaded at runtime based on a user specified configuration file.
- **Reusable Software Components:** While the specific details of V&V vary from application to application, the macro scale algorithms used are relatively consistent, including; mesh refinement studies, using the method of manufactured solutions, sensitivity analysis, uncertainty quantification and error propagation. Many of these algorithms can be, and in some cases already have been (e.g., DAKOTA, MASA), implemented as black-box or near black box solutions. The VnV framework will look to capitalize on this fact by providing a set of robust, near black-box, V&V tools that can be integrated into codes using the VnV runtime test injection system.
- **V&V Testing Efficiency:** Performing a large number of 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. Where possible, the VnV framework will offer functionality for offloading the execution of in-situ tests to external processes. Offloading of the tests to an external server will significantly reduce the run time costs of in-situ V&V with the VnV framework; further increasing its utility in HPC applications.
- **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 in an almost impossible task. The VnV toolkit will include automatic VnV report generation in the form of a server-less html web page that can be viewed in any modern web browser. The entire web page will be built using an extended markdown format with support for standard markdown formating, latex formating, images, videos, self-sorting tables, two dimensional charts, and three dimensional visualization.

Figure 1 provides an example of how developers and end-users will interact with the VnV toolkit. In this case, we show an example of how the toolkit functionality might be implemented in the MOOSE toolchain.

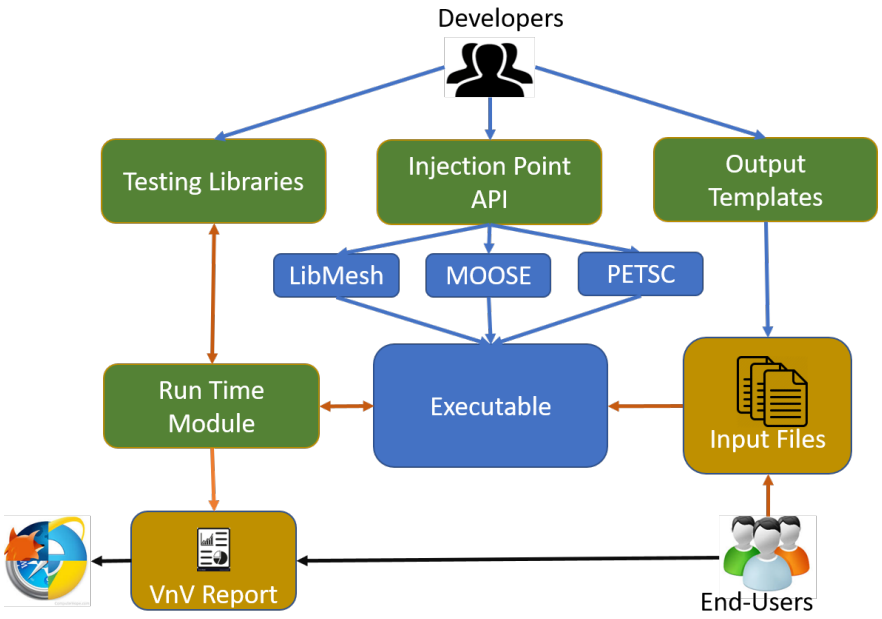

Figure 1: The VnV toolkit development lifecycle. Here, green boxes represent core VnV functionalities. Developer interactions are shown in blue, runtime interactions are shown in orange and post-processing interactions are shown in black.

The first step in the VnV development lifecycle is to specify and describe the injection points. These injection points will be placed at key locations of the code where testing can and should take place. In the Phase II product, inserting an injection point into a code will be as simple as annotating some variables for inspection and calling an injection point function. In addition to adding the code, developers will also complete an output template that will be used in the final V&V reports to describe the state of the simulation each time that injection point is met. The VnV 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 code. The core framework will also include a robust set of general purpose V&V tests. We envision that the developers of a numerical simulation package will ship the library with hard-coded injection points, a set of custom V&V tests and a number of VnV configuration files.

The end-users of the simulations will interact with VnV framework through the configuration files. Through a simple function call, users will be able to pre-generate a VnV configuration file that contains information about every injection point that is contained in any VnV enabled library in the simulation tool-chain. After customizing that file, generating a VnV report is as simple as running the simulation and opening the report in any web browser.

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 user simulations, thereby increasing the usability of the tools for non-expert end-users. The framework will support a full range of in-situ V&V tests with functionality for offloading in-situ tests to external processors. Finally, the package will include support for generating a highly customizable, web-based V&V report. A commercial license will be required to incorporate the VnV toolkit into for profit applications; however, 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

The initial customers of the VnV framework will include the businesses and other institutions (e.g.,ANSYS, Cd-adapco, government labs, universities) that develop large-scale numerical models and simulations. To these customers, RNET will provide the training, support and integration services required to quickly and efficiently integrate the VnV framework into new and existing code bases, as well as contract based services for extending the toolkit to fit certain needs. For these customers, the benefits could include streamlining of the companies internal V&V practices, and an increase in the usability, reliability and confidence in their final product.

The true beneficiaries of the VnV framework are the users of the advanced numerical simulation products. By removing the burden associated with V&V , the VnV framework will ensure erroneous errors do not propagate into final designs. All in all, the VnV framework will afford researchers and engineers with the knowledge required to drive the next generation of technological advancements.

1.3 Phase I and Feasibility Demonstration

The primary goal of the Phase I effort was to demonstrate the feasibility of in-situ end-user verification and validation of numerical simulations. To that end, the project team spent the majority of the Phase I project developing an initial prototype that demonstrates the core functionality of the VnV toolkit.

1.3.1 The VnV Injection Point System

The core functionality of the injection point system was written in C++ with a focus on minimizing the amount of code required to integrate the injection points into existing code bases. Figure 2 shows the three stages of specfiying an injection point; the declaration; the specification; and the final HTML rendering. In this case, the injection point is applied inside a class member function that evaluates a linear function at a particular value of x.

Integration of an injection point into an existing code is a simple, three step process; (1) include the "vv-runtime.h" header file, (2) place injection points in the code and (3) write the injection point specification.

At the core of the system is 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. Tests defined on staged injection points stay in scope across all stages allowing for data collection across multiple code locations. The variable parameters allows the develop to specify which variables can be inspected and tested at each injection point. The phase I prototype uses a combination of string based type checking, static casts to (void*) and C style variadic function calls to achieve this task. To enter a variable, the developer must pass the type and the variable to the macro. When the function is called at runtime, the VnV injection point factory compares the types accepted by the specified tests against the type specified at the injection point. There are several risks associated with this approach, the most important of which is memory corruption from a bad cast. A key goal of the Phase II project will be to develop a set of custom preprocessor directives that use compile time checks to minimize these risks. (see Section 2.2).

The final step in the creation of an injection point is to write the injection point specification. The specification is a YAML file containing the content used to populate the final VnV report (see Figure **??**). The only required parameter in the specification is the name; however, the more information that is entered here, the more informative the final report will be. The content entry represents the text that will be shown in the final report each time the code reaches the associated injection point. This content is specified using

Figure 2: Sample code showing two injection points inserted into a C++ class that represents a linear functions. Top: The source code modifications required to specify an injection point. Middle: The injection point specification written using the custom markdown extension; Bottom: The final rendering of the injection point in the VnV report.

Figure 3: Sample code a custom VnV test for tracking the slope and intersection point of a linear function.

a custom markdown format that supports all standard markdown commands, along with a range of data visualization extensions (described below).

The second facet of the injection point system are the specific V&V tests. The development of the test interface was based on the idea that tests should be loaded at runtime and defined independently of the source code. To achieve this, the test interface was built using a C++ plugin pattern.

The first step in the development of a new VnV test is to create a testing library. The framework includes a library generation script that will automatically build the directory structure and makefiles required to build this library. All that is required of the user for this step is to call this script with a unique name.

Once the library has been initialized, the user can begin to develop individual tests. Writing a test is as simple as implementing the IVVTest Interface. To assist in the development of tests, and to avoid issues with incorrect type-casting, a python based test generation script is created as part of the library initialization step. This script can be used to automatically generate the boiler plate code required to implement the testing interface while also taking care of the required typecasting.

Implementing the IVVTest interface is a three step process. First, the developer must define the names and types of the parameters the test will support. This is completed in the constructor by adding the name and type of certain parameters to the parameters list. The only requirement on parameters is that the names should be unique within the test. To ensure efficiency in HPC settings, the VnV framework uses ADIOS for all data output. Writing data inside a test using ADIOS is a two step process that requires the user to (1) declare the output variables that will be written and (2) write the data. Tests should declare the output variables in the DeclareIO function. Note that it is not a requirement that the output variables be defined in advance, however it is good practice and allows for more efficient handling of the meta-data in ADIOS. The actual testing and writing of the data should occur in the runTest 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.

As with the injection points, the final step in the test creation process is to define the test specification. This specification acts as the link between the data collected during testing and the final report. Figure 3 shows an example specification for a test designed to verify and validate the slope and intersection point of a linear function. The test specification uses the custom extended markdown format to automatically populate and render an interactive line chart representing the linear function. This specification was used to render the “Linear Function Plotter” section shown in Figure **??**

Our vision for the VnV toolkit is that each numerical library will ship with hard-coded injection points and a custom VnV Testing library. The core VnV framework will act as a single interface to these individual libraries, allowing the end-users to build explainable numerical simulations with integrated, in-situ testing at every level of simulation hierarchy.

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. Please see the final report for a full description of the XML format supported by the phase I

prototype.

With the configuration file in hand, performing VnV in a simulation is as simple as including a header file, linking the VnV library and calling the VnV Initialization function.

1.3.2 Automated 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, the chart shown in Figure 2 utilizes the support for generating two dimensional plots using the format. Other functionality supported includes 3D visualization with VTK.js and interactive tables with tabular.js. The extension also includes support for custom post-processing scripts. This allows for infinite possibilities with regard to processing the test outputs. For example, it is possible to run a paraview script that generates VTI files based on the test data.

TODO Figures **??** show screen-shots of a sample VnV report generated using a set of toy testing libraries. The main layout consists of three components; the carousel, the index and the content. The carousel is an optional component that can be used to highlight important results of the simulation. It accepts up to ten pictures, each with its own custom caption. The index and content are generated automatically based on the VnV output file. Each node in the index represents an injection point encountered during the simulation. As such, this index represents a coarse grained view of the simulations call stack. At the top of each injection point section is the injection point content as specified in the injection point specification. Following this is the output of each test completed at the injection point. Last is the list of children. These children represent injection points that were initialized between the first and last step of a staged injection point.

1.3.3 Demonstration in a Moose Application

To demonstrate the utility of the method, the project team placed several injection points in the main function of a MOOSE example "ex01", one injection point in the PETSc Initialize function and one injection point in the PETSc Finalize functions. This is an extremely simple example that did not test the full limits of the new API; however, it did act to verify that the phase I prototype can be used to can be used to perform in-situ verification and validation in a across multiple libraries through a single interface.

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 python based report generation code that automatically creates an interactive server-less web report based on the VnV output.

As will be described in the workplan, the goal of the Phase II project will be to take this initial prototype and extend, harden and optimize it such that it can be efficiently used 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.

In Phase II, RNET Technologies and ORNL will pursue the following objectives:

1. {Harden and extend the core VnV functionality developed during phase I. In particular, the phase II effort will look to determine the optimal approach for implementing the run time test injection system such that the risks associated with memory corruption are minimized. }

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.}
3. {Minimize execution times through job based parallelism. This key issue to address here will be to develop a mechanism for offloading tests to external processors such that the data transfer is faster than simply running the test as-is. 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 further equip the users with the tools required to robustly perform end-user V&V .}
5. {Demonstrate the value of the VnV framework as a component in the NEAMS toolchain and integrate it into the NEAMS workbench. By showing the toolkit can be used in the NEAMS toolchain, and in particular, MOOSE, libMesh and PETSc, we will demonstrate true potential of the product in libraries that are already considered cutting edge across the industry. Integration into these tools will answer the NEAMS call for tools that support end-user verification of numerical simulations in its core tools and 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 satisfy the objectives outlined in the previous section.

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 type-casts. In a trusted environment, this is an efficient (void* casts are almost free) and portable (it uses low level C functionality supported by all C/C++ compilers) approach. However, there are two main weaknesses with this approach:

- **String based pointer casts:** The injection point system developed in Phase I requires the developer to provide a string that describes the type for each variable available at an injection point. Under the hood, the injection point system using string comparisons to ensure compatibility between test parameters and injection point variables. The key issue with this approach is that the strings specified at each injection point cannot be verified during compilation. This will causes major 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.

In Phase II, the project team will investigate and develop an annotation based system for defining injection points. This annotation system will be far more dynamic, providing 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 of variables. The annotation system will also provide a mechanism for suggesting default tests

to be run at each injection point, and will 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. In particular, Clangs simple API for defining custom Pragma routines, combined with its robust API for walking the Abstract syntax tree of the code make it the perfect choice for developing this system. Clang has seen wide scale uptake across the software development world and, in recent years, has emerged as a realistic competitor to the ubiquitous GCC compiler suite.

2.2.2 Efficient Statistical Comparisons in Distributed Settings

One of the major benefits of the VnV framework is that it allows for in-situ testing and analysis in distributed systems. This type of in-situ analysis is particularly useful for analyzing and comparing data stored in distributed arrays. For example, VnV tests could be used to assert that all the elements in an array are positive, or that all the elements in an array are the same as the elements in another array within some tolerance. From the perspective of V&V , such tests would allow for in-situ comparisons with experimental data and for fine-grained regression testing. To that end, a large portion of the Phase II project will be devoted to developing efficient methods for analyzing, asserting and comparing data stored in distributed arrays.

Some information about the global distribution of the data is needed to act of distributed data. The naive approach to determining the data partition is to explicitly form the global 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 [**?**]. 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 such as forming a hash based distributed directory [**?**] and the assumed partition algorithm [**?**].

Once the intra-processor communication patterns have been determined, the focus will shift to developing efficient algorithms for comparing and analyzing distributed data. 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). The testing suite will also include several tools for comparing data stored in distributed arrays with data stored on file and data in other arrays.

2.2.3 Reducing Run-times with Test Offloading and Job Parallelism

Performing a large number of 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 have built in support for offloading 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 the optimal architecture (e.g., an 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 via the ADIOS2 write/read API. In HPC environments, SST uses the RDMA interconnects to ensure fast transfer of data between HPC applications; however, socket based connections are also supported.

The VnV toolkit will use the SST engine to distribute the required testing data to a VnV Testing Manager. The role of the test manager will be to determine, based on the data provided, the best available resource for running the given test and to launch the required executables. The test manager will be a C++ executable that makes use of 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, the test manager will use the PTP platforms extensive support for integrating with remote job schedulers to launch tests on the best available resource.

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

Mesh refinement studies, uncertainty quantification and sensitivity analysis are all essential components of a robust V&V regimen. To that end, the Phase II effort will include the development of V&V tests that integrate third party tools to complete these tests.

The UQ and SA tests will be developed using DAKOTA. DAKOTA provides a set of black-box tools for performing parameter optimization, UQ and SA. DAKOTA provides two interfaces for integrating these tools into applications; a black-box approach that uses preprocessors, post-processors and the file system to complete the tests; and a library API for direct, hard-coded testing. The Phase II effort will include the development of an custom VnV testing library for direct integration with the DAKOTA library API. This will include the development of a flexible interface for setting up and running the DAKOTA tests, and the development of the test specification files for displaying the test results in an informative and interactive way.

Support for Mesh refinement will be developed as part a separate plug-able VnV testing library. 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

The next step in the Phase II project will be to integrate the VnV framework into the NEAMS toolchain. 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 answer the solicitations call for tools that support end-user verification of numerical solutions in the NEAMS tools.

Integration into NEAMS toolchain 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 obvious 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 project team demonstrated the ability to insert injection points into MOOSE code during Phase I.
- 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 be completed by developing a custom MOOSE "Action" that sets up the core VnV testing functionality.
- 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. To do this, we will create a lightweight executable that loads external test libraries and extracts the appropriate parameter specifications in the "SON" format supported by workbench.

- The last step will be the development of an interface for editing the YAML specification files used to generate the final report and an interface for viewing the final reports. The GUI will be written using standard HTML and Javascript and then integrated into the workbench using the QT QWebEngineView component. The QWebEngineView component allows for interactions with the core QT window (i.e., open, save, copy, paste, etc.) however, it is important to note that this approach will not allow for a truly "native" experience within the workbench. The decision to develop in JS and HTML rather than native QT was made to ensure the final product, the VnV framework, is consistent, and can be used in applications outside of the NEAMS lines of tools. Every effort will be made to ensure the GUI conforms to the NEAMS workbench standards and specifications.

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 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 its collaborators. Figure 4 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.

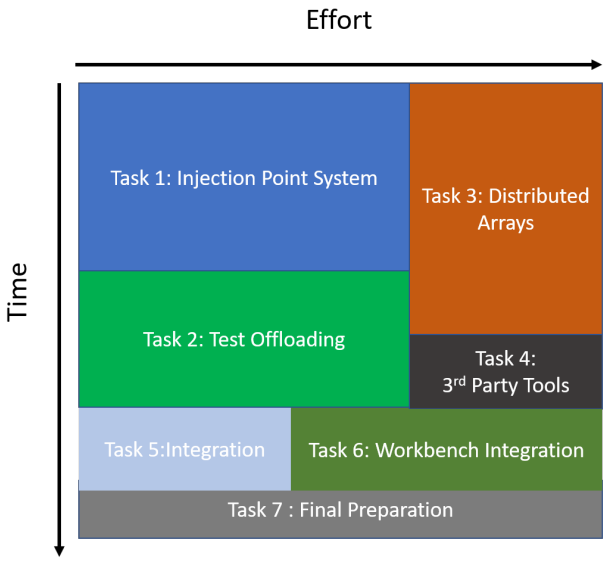

Figure 4: Overview of task dependencies and timeline.

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 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 **??**. 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 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 decompoosition to the individual tests. This will involve robust analysis of three aapproaches 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 real applications, including the NEAMS Tools

{ 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 reimplement 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 for the MOOSE based configuration file specification developed in the previous task. 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 will be developing an interface for customizing and viewing the final VV report. As part of the Phase I effort, the project team demonstrated viewing the final V&V report in a QT WebView component. The workbench is also built using QT, hence we do not expect to many difficulties on that front. Instead the key objective will be to develop the mechanisms for displaying customizations made the the final report in real-time within the NEAMS workbench. } { 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, tutorials, etc.), extending the custom VnV markdown specification and adding support for generating reports that conform to industry specific standards.}

{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.

Gregory Watson, PhD, is a Senior Research Scientist in the Computer Science Research Group at Oak Ridge National Laboratory. Dr. Watsons 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 (Dr. 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 [**?**] 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.