Response to the Request for Information: Open Source Software Development Acceleration, NNSA/ASCI

This response has been assembled by a collaboration of academic researchers. It describes specific areas and capabilities for application performance analysis tools where development could be accelerated through a government funded open source development effort. This effort would positively impact the time-to-solution of ASCI Tri-Laboratory application development efforts. The plan put forth here proposes to further develop and integrate into a unified framework technologies already in use within both the NNSA and the DOE Office of Science. The response includes a plan for future productization of the proposed open source technologies. The response also includes descriptions of anticipated integration with the compiler area, interactions with the HPC cluster distribution and resource management areas, and use of and contributions to the Open Source system test/certification facility.

Academic researchers:

Jack Dongarra, Shirley Moore, Philip Mucci, and Daniel Terpstra, University of Tennessee
Allen Malony and Sameer Shende, University of Oregon
Jeffrey Hollingsworth, University of Maryland
Barton Miller, University of Wisconsin
Daniel Reed and Celso Mendes, University of Illinois
Allan Snavely, San Diego Supercomputer Center

We propose the definition and partial implementation of a portable open source architecture for performance data acquisition and analysis. The proposed work will result in a portable, open source framework for instrumentation of applications and measurement and analysis of performance data. The framework architecture will be structured, to enhance separation of functionality, as well as integrated, using well-defined interfaces between components. The architecture will be designed for scalability and efficiency, thus extensible to systems with thousands of processors. The framework will be populated with proven, working component tools at each layer. However, because each interface will be well documented with clear functional isolation, the framework will be extensible and augmentable with emerging new tools via common APIs and data formats. The framework will satisfy two goals: (1) to provide a coherent working suite of tools and (2) to provide tool researchers a firm foundation on which to base focused tool efforts.

The architecture has four primary layers: (1) instrumentation, (2) measurement, (3) CCT (command, control, and transport), and (4) analysis. The instrumentation and measurement layers are connected by a common measurement API so that the

performance events identified for observation are decoupled from the particular technology used for instrumentation. The measurement and analysis layers are *logically* connected by an infrastructure for data management so that data can be stored, shared, and operated upon by standard means. Layers are vertically integrated by a collection of well-defined events and contextual hooks that provide a mapping of low-level data to high (user) level information. Between horizontal components and vertical layers is a virtual layer, CCT, that provides efficient dissemination of commands and control operations, and collection and filtering of data from a large number of execution nodes. The proposed architecture is shown in Figure 1.

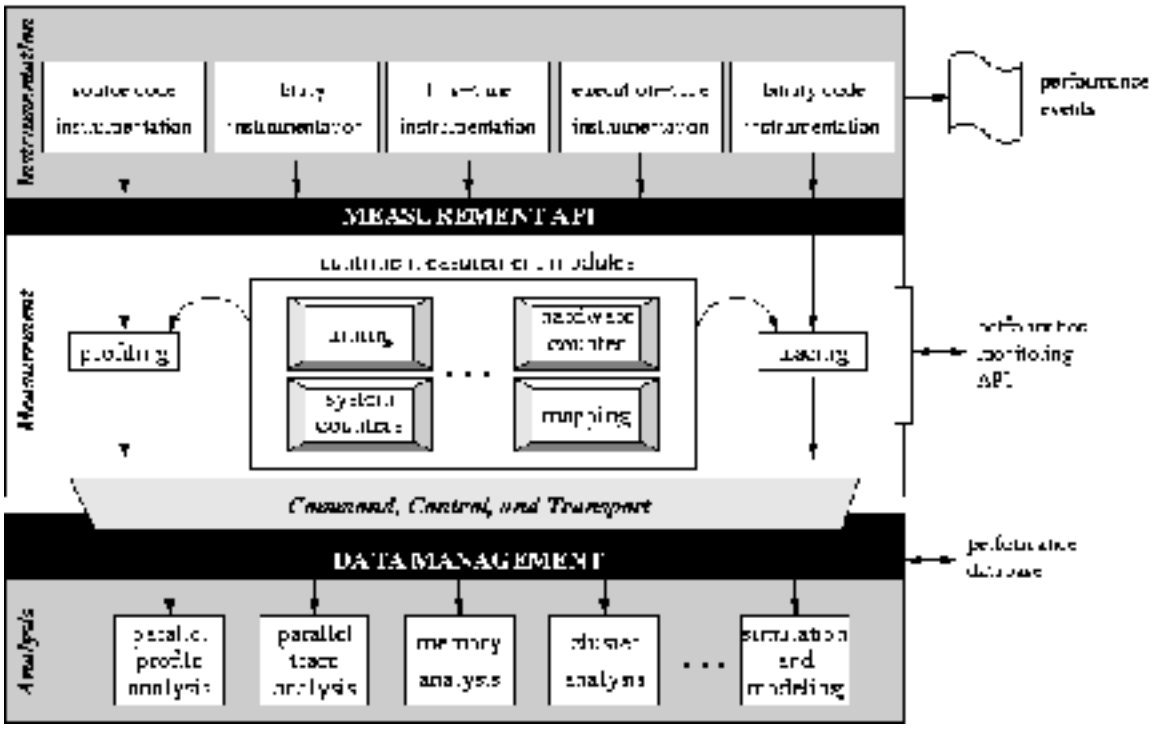

Figure 1: Architecture for Performance Data Acquisition and Analysis

Instrumentation Layer

The instrumentation layer will provide tools for the automated insertion of measurement functions with respect to instrumentation points. It will define these points compatible with user semantics, thus answering "what are meaningful instrumentation points?" These points will range from the low-level (e.g., instructions, basic blocks or memory locations) to the high-level (e.g., source-code constructs such as loops, functions, and objects). The instrumentation point definitions will be made available to high-level tools via the measurement API. Instrumentation techniques to be supported in the framework include source code, library, link-time, binary code, and execution-time instrumentation. This layer will enable dynamic instrumentation for attachment,

detachment, and re-attachment to running programs without recompilation or re-execution.

Measurement API

A measurement API will provide for instrumentation mechanisms to target a common measurement infrastructure. The interface will be extensible to enable the definition and integration of new measurement functionality. (Pre-defined measurement functions are provided at the next layer).

Measurement Layer

The measurement layer will provide a rich toolbox of measurement operations and data structures to capture runtime information about performance events. Common profiling (i.e., statistical summary) and tracing (i.e., event recording) functionality will be provided here. Hardware performance monitoring via the PAPI portable interface will be supported. The measurement infrastructure will be organized as a set of runtime modules configurable statically at compile time or dynamically loaded. The measurement layer is inherently extensible with the addition of new modules. It is also at this layer where options for performance data access are supported. Both end-of-execution and various forms of online access (i.e., real-time performance monitoring) will be supported.

Data Management Interface

This interface lies between the measurement and analysis layers. It defines the API between these but, for scalability reasons, parts of the actual functionality may reside in the virtual CCT layer. The data management interface will provide standardized performance data formats for outputs of measurement functions appropriate for both low and high-level performance features ranging from raw instruction counts to summarized instruction patterns to results of highly summarized on-line analysis. It will also supply data handling/filtering functions to reduce size of outputs of measurement functions including the following:

- Performance data size reduction via online analysis and summarization along with standard formats for summaries of performance data
- Performance data size reduction via sampling along with standard formats for sampled data (i.e., interval, frequency).
- Performance data size reduction via pattern detection and other high-order performance feature extraction, along with standard formats for instruction patterns. Functions placed in the data-acquisition hierarchy primarily for the purpose of reducing data size or manageability are here classified as being in the data management interface. When part of size reduction involves point-of

acquisition analysis, these could just as reasonably be classified in the next layer up (analysis).

Data management will further supply standardized performance data formats for outputs of data handling/filtering functions and a toolset for common data handling/filtering/compacting functions with an API allowing definition of new data handling/filtering functions.

A performance database schema will be developed for storing raw and processed measurement output and analysis results and to allow for rapid retrieval to enable decision-making based on history.

Command, Control, and Transport (CCT) Layer

The CCT layer will provide the means to scale this framework to 1000's of nodes. It consists of a facility to construct an overlay network to support multicast of command and control information to execution nodes and the ability to support distributed processing of data streams as they are collected. This processing is done in data *filters*, which are distributed throughout the overlay network. While common filters will be provided, tool builders will be able to construct new ones and dynamically load them into the CCT layer.

Analysis Layer

The analysis layer will provide a rich toolset for building data analysis functions to interpret the output of the measurement system, with functionality including performance modeling, simulation, and visualization. It will also include an API allowing for definition of new data analysis functions and for plug-and-play with new and existing analysis functions.

Vertical Integrators

The scalable infrastructure will allow for efficient collection, reduction, transport, and analysis of performance data from thousands of nodes, connecting measurement, filtering/compacting, analysis/modeling/visualization functions in a flexible, high-performing fashion.

Contextual information for vertical integration across all the layers will allow users to tie results of measurement and analysis back to application abstractions (as for example to be able to correlate instruction counts to source-code constructs)

3.1 Current Status

Instrumentation Layer

The Program Database Toolkit (PDT) has been developed during the last five years to provide a framework for building source analysis tools. It uses commercial-grade parsers for C, C++, Fortran 77/90/95, and produces a program database with a C++ class library (DUCTAPE) for accessing the PDB. PDT has been ported to almost all platforms where
TAU is available. TAU (see Measurement API, below) uses PDT to implement automatic source instrumentation tools. LANL uses PDT to implement automatic interface generation for language interoperability, as part of the CCA project. TAU developers have been working closely with Bernd Mohr (Research Centre Jülich) on PDT. Mohr has an OpenMP source instrumenter, OPARI, that is used in the TAU project. When PDT obtains statement-level analysis capabilities (being worked on now), OPARI may be rewritten to use PDT. PDT is an example of project that has received very little funding to date.

Dyninst has been developed to provide a platform independent framework for runtime instrumentation of programs. Dyninst capabilities include binary analysis (including control flow graph extraction and natural loop identification) and runtime code insertion (at selected procedures, loops or instructions). Dyninst currently runs on Alpha, MIPS, Power, SPARC, and IA-32 (x86) processors. A port to IA-64 is underway, with a pre-release version scheduled to be available at the end of 3Q03. Programs written in C, C++ and Fortran can be instrumented. Currently support for some C++ features (inheritance and template classes) is functional, but somewhat awkward to use. In addition to runtime changes to programs, Dyninst can also be used as a binary editing tool (which is often desirable for tool use in a batch environment). Dyninst is based on a single node instrumentation model; however, tools such as Paradyn and DPCL have been built which provide multi-node instrumentation using Dyninst for intra-node instrumentation.

SvPablo is a performance analysis toolkit that contains instrumenting parsers for C and Fortran applications (C++ support is under integration via ROSE). Using those parsers, one can instrument procedure calls and outer loops in the source code of an application. During execution, SvPablo's data capture library maintains counts, durations and other metrics obtained from hardware performance counters for each instrumented fragment. Instead of producing a regular trace like other tools, SvPablo's library keeps a summary of the captured metrics. Hence, instrumented programs can run for hours or days, on thousands or processors, without producing excessive performance data.

Measurement API

The TAU project has gained significant portability and robustness in performance measurement capability through the use of a common target for performance

instrumentation. The current interface includes support for standard and user-defined events, performance mapping, and performance data access. A common API allows different compatible libraries to be used. The TAU measurement library can be configured to support profiling and/or tracing, call path profiling, counter measurements, use of different timers, and several other user-determined features. The measurement system (API and library) works across all major parallel platforms and for all major programming languages: C, C++, Fortran 77/90/95, Java, Python. It can be accessed from different instrumentation levels and can utilize other technology such as Dyninst and PAPI. An abstract computational model allows the measurement API and library to be applied to all parallel execution models.

Measurement Layer

PAPI provides a portable interface to hardware performance counters on modern microprocessors. These counters exist as a small set of registers that count *events,* which are occurrences of specific signals and states related to the processor's function. Monitoring these events has a number of uses in application performance analysis and optimization. PAPI has been incorporated into several third-party commercial and research performance analysis tools, including SvPablo, TAU, and Dynaprof, and includes bindings for C, C++, and FORTRAN for direct application in end user codes. PAPI has become a *de facto* industry standard and is sponsored by the Parallel Tools Consortium (http://www.ptools.org/). PAPI is currently supported on a broad range of architectures, including the IBM POWER series, HP Alpha, Intel/AMD Linux, MIPS, Sun, Cray, and others. Ongoing development effort is invested to port PAPI to newer architectures such as the Cray X1, AMD Opteron, and others. Simultaneously, a major development revision is underway to streamline and optimize the PAPI hardware independent layer to both incorporate additional features available on newer architectures and to simplify the process of porting to additional architectures.

Data Management Interface

A first version of the Performance DataBase Framework (PerfDBF, soon to be released) has been developed at University of Oregon with an ASCI Level 3 grant during the past year. PerfDBF uses PostgresSQL (or MySQL) to store multi-experiment parallel profile data. PerfDBF can currently input TAU profiles, but plans are to build other profile data readers. An API has been implemented for building performance analysis tools that offers a higher-level interface for querying the database. Of course, the tool can use SQL commands directly. The TAU project is developing an analysis toolkit above this layer to provide commonly used analysis functions. For example, the TAU ParaProf tool currently can use a PerfDBF-created profile database as an input source.

SvPablo, a graphical performance browser developed at Illinois, represents the third generation of the Pablo tools for performance data capture, analysis and visualization in

parallel applications. As such, the toolkit reflects experience obtained on multiple platforms and environments. Under the scope of the NSF Alliance's Performance Expedition, SvPablo performance data has been imported to the Prophesy system, an infrastructure developed at Northwestern University for analysis and modeling of performance in parallel and distributed applications. Prophesy is based on the Postgres database, and supports both storing observed data and predicting performance for other configurations of the same application/platform pair. In the current version, this integrated system can only handle the durations of application fragments, but the plan is to extend this support to the other metrics produced by the SvPablo data capture library, such as data from hardware performance counters and rates/volumes from MPI communication.

Command, Control, and Transport (CCT) Layer

Runtime tools are crucial to program development. In desktop environments, we take tools for granted. In the Grid, it is difficult to find tools because of the complex interactions between applications, operating system and layers of job scheduling/management software. Therefore each runtime tool must be ported to run under each job management system; for *m* tools and *n* environments, the problem becomes an *m***n* effort, rather than *m*+*n*. The consequence is a paucity of tools in distributed and Grid computing environments. In response, we analyzed several scheduling environments and run-time tools to better understand their interactions. We isolated what we believe are the essential interactions between tools, schedulers, resource manager, and applications. We have proposed a standard, called the Tool Daemon Protocol, which codifies these interactions and provides the necessary communication functions. We implemented a pilot version of this library and experimented with Parador, a prototype using the Paradyn Performance tools under Condor. The TDP effort has been funded to date as a background effort from the Paradyn and Condor projects. While there is significant interest in this work, distribution of the code and wider porting has been limited pending funding directed specifically at this effort.

MRNet is a software-based multicast/reduction network for building scalable performance tools, system administration tools, and Grid middleware. MRNet supports multiple simultaneous, asynchronous collective communication operations. It is flexible, allowing tool builders to tailor its process network topology to suit their tool's requirements and the underlying system's capabilities. MRNet is extensible, providing an open interface to allow tool builders to incorporate custom data reductions to augment its collection of built-in reductions. MRNet has been used for more than multicast and simple data reductions; current applications include more esoteric reductions such as custom histogram (binning) and clock skew detection s. MRNet has scaled gracefully in initial tests on up to 1000 tool back-end processes.

Analysis Layer

The TAU project is developing a C++ performance visualization library and associated toolkit that supports commonly used 2-D and 3-D performance displays. The library will provide high-level classes that can be inherited from for extending the set of display types. The toolkit will include visualization GUIs and other support for building new display tools. The goal is to build the library in C++ with an abstract interface to a performance graphics library, currently being written in OpenGL.

The TAU ParaProf tool is a parallel profile analysis system being developed to read parallel profile data from multiple sources and across multiple experiments. It has a modular component architecture that can be extended with new analysis and visualization modules. It can handle large-scale parallel profiles (thousands of processes). It is implemented in Java using Swing and is highly portable. Significant attention has been paid to code and memory optimization to maintain high interactivity.

The emphasis in SvPablo's design was to stress correlation between observed performance and application source code. In its current version, SvPablo contains a set of hierarchical displays that enable the user to easily correlate all the performance data captured during application execution with the corresponding line in the application's source code. This GUI is based on Motif, and contains a variety of color-coded, clickable fields that both immediately drive user's attention to critical parts of the code and at the same time allow the user to check more detailed information about the performance data. Current work focuses on support for additional analysis capability and corresponding visualization, in the form of compact application signatures that will be generated by SvPablo's instrumentation library. These signatures are a compressed representation of a traditional trace, and still contain the salient dynamical features of the execution.

The PAPI project has started developing a generic and easy-to-use presentation component called CUBE to display a wide variety of hierarchical performance data. CUBE will implement a performance algebra providing the ability to perform arithmetic operations of different CUBE data sets, such as mean or difference, and to display the result like the original data sets. CUBE might also serve as a "draft" of a "standard" profiling data representation.

The Performance Modeling and Characterization (PMaC) Laboratory tools Metasim Tracer and Metasim Convolver are components of a toolset for the extraction of application performance features, automated production of performance models, and evaluation of parameterized performance models via statistical simulation. Performance modeling frameworks based on the toolset have been shown effective in explaining the performance of applications on several current HPC systems and predicting their performance on future systems.

3.2 End State of the Project

If the project were fully funded, the result would be an integrated and extensible performance tool infrastructure that provides the components for building flexible performance analysis tools targeted to meet specific user needs. The communication infrastructure would enable efficient and scalable communication among different tool components, between tools and application processes, and between tools and the runtime system. The standardized measurement API would facilitate insertion of instrumentation code for different purposes and at different stages (e.g., source code modification, compile-time, link-time, run-time) in a standard manner. The data management interface would provide the necessary information about the program source code to enable performance data to be correlated with source code constructs and would provide library routines for accessing and manipulating this information. The performance database would standardize the format for profile data and provide a standard API for accessing this data from relational databases. Standardized GUI components in the analysis layer would enable scalable visualization of parallel profile data for very large numbers of processors and facilitate correlation of profile data with source code constructs.

This effort would impact the HPTC community by providing a firm foundation on which to more quickly and more easily build effective performance analysis tools. This project would impact the ASCI Tri-Laboratory community in particular by addressing issues of scalability at the instrumentation, run-time communication, and data analysis stages of performance tool operation, and issues of time-tractability in the formation and evaluation of performance models.

3.3 Collaboration

The project would be managed by assigning each area to a lead institution that would be responsible for setting sub-project milestones and assigning tasks to appropriate team members to achieve sub-project goals. For example, University of Tennessee would be responsible for the hardware performance monitoring area, and University of Oregon would be responsible for the standard instrumentation API. The source code would be managed in network-accessible CVS repositories, as is currently done in the Paradyn/Dyninst, PAPI and SvPablo projects. Collaboration with tool developers in the Tri-Laboratory community has already occurred in the PAPI, TAU, and Metasim projects and is expected to continue. For example, John May at LLNL developed code for software multiplexing of hardware counters that has been incorporated into the PAPI source code. Phil Mucci, the lead developer for the PAPI project, has contributed code to Tri-Lab tools projects. Allan Snavely works closely with Jeff Vetter and Bronis de Supinski of LLNL in their joint research for the extraction of performance-meaningful

attributes of memory, communication, and I/O access patterns, and they co-advise two UCSD PhD students in this area.

The researchers on this project intend to select, generalize and extend where necessary, and combine the best technologies from their respective projects into an integrated tool infrastructure. Thus, this project would provide a unique opportunity for a focused effort to develop a coherent framework for large-scale performance analysis.

3.4 Accelerated Development

In addition to the component-specific accelerated development described below, the main efforts to be accelerated are the definition of tool interface standards and integration of the components into a coherent tool infrastructure. One need only look at the impact of the National Middleware Initiative (NMI) in packaging, documenting and hardening Grid software to see the value of coordination and integration. Similar success stories exist for cluster software.

Support for participation in standards-promoting organizations such as the Parallel Tools Consortium will help with standardization of tool interfaces. Involvement of vendors in this Open Source effort will motivate vendor implementation of the standards. Without direct monetary support for integration, tool development would continue along the current paths with some pair-wise integration but without a coordinated effort at multi-way integration.

Dyninst: Dyninst has been incorporated into several other available tools, including DPCL (originally from IBM and now an Open Source project) and Dynaprof. The focus of this project's efforts with Dyninst would be concentrated in the areas of making Dyninst more robust and scalable (executables in the size of tens of megabytes can be slow to startup) and in integration with other tools described in this proposal.

PAPI: OSSODA support would ensure that PAPI could be ported in a timely fashion to platforms of interest to the NNSA and ASCI communities, including ASCI Purple, Red Storm and others. It would also provide enhanced opportunities to incorporate advanced features such as multi-way multiplexing for better hardware utilization; parallel overflow and profile support; data-tagged event monitoring; and more comprehensive support of events native to specific platforms. Further, greater attention could be paid to software design changes that would move the PAPI project out of the research realm and into the realm of a robust commercial-quality product, with documentation and design enhancements to make it more feasible for commercial vendors to develop and provide their own interfaces to the PAPI library for new hardware architectures.

TDP: The OSSODA support would offer the opportunity to take the TDP library and fully integrate it with several process management (scheduling) environments. This

level of support will provide the critical mass to make it a desirable porting target for a variety of runtime tools. While TDP is likely to *eventually* become widely accepted, we have the chance to substantially accelerate that acceptance. This acceptance will provide the ability for tool writers in both big *and* small projects to deploy their tools for real applications in real environments; it will no longer be just the providence of big projects with staffs that can handle the porting efforts.

MRNet: The MRNet libraries are currently in limited distribution and development for cluster and SP environments. The OSSODA support would offer several crucial advantages: (1) porting additionally process management environments, (2) addressing technical issues relating to MRNet as Grid middleware (e.g., security and interoperability issues), (3) support and extensive testing.

TAU: The TAU measurement API would be extended to provide the following:

- Per-event measurement specification
- Improved dynamic measurement control
- Better runtime performance data access (both application-level access and external access)
- New parallel tracing library with additional runtime statistics options

SvPablo: While we continue to add additional features in SvPablo using funds from other sources, we plan to use OSSODA support to accelerate the integration between our future SvPablo versions and the various other tools described in this document. One of the forms for this integration would be the export of SvPablo-produced performance data. We will create conversion mechanisms allowing those other tools to access the SDDF files with performance data captured by SvPablo. Similarly, we intend to use OSSODA support to promote this tool interoperability in a portable fashion, covering all the platforms of interest to ASCI. As an example, we have developed in the past a prototype version of a package that produced dynamic instrumentation in SvPablo using Dyninst,
under Solaris systems. With the new funding, we plan to update that functionality with the current Dyninst version, and extend it to other platforms where Dyninst is available. Likewise, we plan to continue using PAPI's new versions to access hardware counter data in all the platforms where a PAPI port exists.

MetaSim: Support would accelerate porting MetaSim Tracer to run atop Dyninst; it would then be hardened and made modular via an API so that it can be called from higher-level performance observation frameworks. The result would be memory access pattern detection capability on all the platforms where Dyninst runs with the ability to enhance trace performance information from hardware counters with additional information about memory stride and reuse distance at the basic-block level and with tags back to the source code (something that is not available from raw counters, nor supplied at the high level today). Additionally MetaSim Convolver would be made

modular and integrated into a performance observation framework based on capabilities of TAU and SvPablo. The result would be performance modeling capability so that on could both observe factors influencing performance on existing machines *and* play "what-if" to explore performance implications of varying attributes of the underlying machine (such as latencies and bandwidths of the memory subsystem or communication fabric) or the source code (such as better blocking for cache in loops or sending fewer, larger messages in functions or loops).

3. 5 Testing and Integration Strategy

A successful testing and certification program for a coordinated Open Source initiative must be built on a comprehensive repository of test problems. Test problems should range in sizes from small kernels to full-scale applications and, along with testing for other areas, should be designed to test correctness, efficiency, scalability, robustness, interoperability and easy of use of compilers and debugging and performance analysis tools. We do not consider the overall construction of the test problem repository to be part of our effort, but we expect to contribute test programs to this repository (e.g., validation benchmarks for hardware performance data, example programs for performance instrumentation with different parallel programming models and using different instrumentation strategies).

The testing strategy for the software proposed in this project is to develop comprehensive test suites for each of the component areas, and for pair-wise and multi-way integration of the components, and use automated multi-site daily regression tests. Currently, the Dyninst/Paradyn project uses a suite of daily regression tests run on 11 different systems at the Universities of Maryland and Wisconsin.

In addition, we plan to have a set of beta testers who collectively have access to the targeted set of platforms. These beta testers would run the test suites as well as test beta releases of the components with additional test programs and applications (both from the test problem repository and from their own applications) whenever possible. This testing strategy would be similar to that used for the PAPI project, which has a suite of test programs (which could use improvement however) and an enthusiastic cadre of beta testers. A second stage of beta testing, after the bugs reported in the first stage have been fixed, would take place at scale on the system test and certification facility described in this RFI.

3.6 Technology Productization Strategy

Our goal is to produce an interoperable collection of tools and associated framework to support performance measurement and evaluation on the ASCI computing platforms.

For this effort to have long-term returns, the tools and framework should live beyond the time frame of this proposal.

Current State of Affairs

The current state of available performance tools has demonstrated clearly that a new approach is needed. This approach must encompass integration and support. Integration is crucial, and we have addressed that topic previously in this document. Note that specific funding for this integration effort will make an interoperable collection of tools possible.

Support is a more complex issue. As a result of economic realities, there are few examples of useful tools available on a variety of platforms. We first discuss the reasons for this situation and then propose a possible direction.

Machine vendors have nice point-solution successes for individual platforms but they no incentives for porting their tools to other platforms. After all, why would a vendor want to make another vendor's platform more desirable and useful? While this strategy is self-defeating, it is difficult for vendors to escape this mind-set. Note that as primarily hardware vendors, this high-end software market is too small to be of interest in and of itself.

Tool vendors, typically smaller software companies such as Pallas, have some cross-platform successes. However these companies have significant investments in their software, and their software is their primary intellectual asset. Creating open source versions of their tools is typically considered a bad idea, giving away their most valuable assets. The small size of these companies makes it difficult for them to keep up with both porting and new technology developments. Totalview is an interesting counterexample, which was only make possible by a continuous infusion of government money from a variety of agencies and from a few companies.

The open source community, as exemplified by GNU, develops sophisticate software on a wide variety of platforms. Individuals, either with or without the official support of their employers, contribute their time to produce and support this software. Note that this software is "big market" software, such as the operating system or compilers. As such, it involves an extremely large user and developer communities worldwide. Performance tools, especially targeted at high-end machines do not fit this profile because they have significantly smaller communities.

Consortiums such as the X.Org produce widely distributed open source systems. X.Org distributes a well-regarded version of the X11 window system for many different platforms. The funding model is buy-in support from member organizations such as

Compaq, IBM, HP, SGI and Sun. Members pay from $15,000 to $50,000 per year to support this effort (with lower payments for Associate Members). Note that there is also a GNU-like team, called XFree86, working along with X.Org.

Proposal for Long Term Support

We propose a three-pronged approach to establish wider acceptance and long-term support of the software that we produce. This approach is not unique to performance tools and might be adapted by other related ASCI open source software efforts. We propose to establish a High-end Performance Tool (HPT) Consortium. This consortium would blend three components, similar to X.Org, but with medium term funding for core activities:

- *Open source independent developers*: Many of these developers would come from the groups participating in this proposal, but we also expect participation from software developers in industry and the government labs.

- *Consortium members*: There are several vendors with large stakes in high-performance computing. These vendors may see a modest membership fee as a cheap path to supporting tool availability. If we were able to attract a steady-state membership size of a half dozen, this would provide a solid core to help fund a web site, coordination of distribution, documentation, and publicity.

- *Government support:* To guarantee medium-term availability of the performance tools, a funded team of integrators and testers would provide continuity and reliability to our efforts. It would be the job of the team funded by this support to ensure that new developments are uniformly available on all platforms and all changes are tested for all configurations. These tasks are extremely difficult outside a core group. This level of software quality could be supported (with the above components) at the cost of three to five FTE's per year. The FTE's could be centralized at one of the participating sites or distributed. Note that when we say "medium term", we mean the five-year period following the end of this supported research (years 4 through 8). The hope is that after five years, the tools would mature to the point that they have sufficient users and developers so that government support is no longer needed.

Summary

The funding of this proposal would provide the impetus to develop core group of tools for performance measurement and evaluation. This funding is the necessary key to start this open source effort. The medium-term support, combined with a well-organized consortium, could establish the long-term viability of this software for the ASCI platforms. We have previously had close collaborations with industry (e.g., with IBM,

Intel, HP, Cray) on projects such as PAPI and Dyninst, and we expect collaborations with these companies to continue and to encourage their membership in the proposed consortium.

3.7 Project Budget

We propose to fund this effort over three years at a total of $5 million for the total period. The funding would be divided between the six participating organizations.

Year 1: During the first year, inter-component interfaces will be defined and robust versions of the components will be released, conforming to these new interfaces. The initial version of the components will be targeted at two major computing platforms of interest (to be decided, based on consultation with DOE). These components will be the first product of the HPT Consortium.

Year 2: The second year will start with extensive intercomponent testing and establishment of the tool framework. The second year will also start development of end-user tool functionality. Inter-component interfaces will be re-evaluated at this stage and adjusted as needed. At least one additional computing platform will be selected at this time. The HPT Consortium will start outreach to establish commercial members and sponsors.

Year 3: The third year will produce an initial set of end-user tools based on our components and framework. During this time, we will test these tools with applications selected under consultation with the Tri-Labs. During the last six months, we will work with Consortium members to develop commercial transitions of the framework and tools. This last stage will also include seminars and tutorials for the HPC community at large.

3.8 Intellectual Property and Open Source License

All tools and components developed under this funding will be available under a uniform open source license, modeled after that used by NCSA.

10.4 Area specific information

The systems targeted by the tool infrastructure are current and future ASCI systems, including AIX/POWER3/4/5, Tru64/Alpha, IRIX MIPS, Linux/IA-32, and Linux/Opteron. Additional platforms will be supported as resources allow. The future ASCI Red Storm and BlueGene/L platforms will be supported.

Languages that will be supported include Fortran 77/9x/200x, C, and C++, as well as mixed language programs. Parallel programming models that will be supported include MPI, OpenMP, Pthreads, and mixed mode (e.g., MPI/OpenMP).

The Red Hat Linux distribution will be used for development. The infrastructure will be updated to the most recent stable Linux release on an ongoing basis.

The NNSA/ASCI Open Source initiative provides a unique opportunity for integration of the compiler and tools areas. Higher levels of compiler optimization apply increasingly complex transformations and optimizations to user source code. The resulting executable code can be difficult to map back to the original source code, limiting the effectiveness of source-level debuggers and performance analysis tools. A high performance compiler should provide source-level tools with detailed information about the transformations and optimizations that have been applied, using a standard interface to do so. The compiler can assist with gathering performance data through instrumentation of the generated code. Being able to specify collection of various levels of profiling and tracing data using a simple set of compiler options would improve ease of use of performance tools. In the other direction, performance data can be fed back into the compiler to provide more effective optimizations in the next development cycle.

Our work would benefit from integration with the open source compilers proposed under this program by linking of the source code to performance tools that need program-level information for instrumentation, analysis, and results display. This would be done by bridging the compiler front-end technology to PDT's program database. Such a linkage has already been demonstrated with PDT's own integrated parsers. Close cooperation with compiler efforts will guarantee that open source parsers and an XIR will function well with PDT, ensuring a evolutionary path for existing and to-be-developed PDT-based program and performance analysis tools. As an example of this approach, University of Oregon is currently working with the University of Houston on the translation of Open64 WHIRL to PDB format. In addition, Code Sourcery, LLC has shown interest in this approach as it relates to their proposed open source compiler activities.

Another area with which interaction would be beneficial is the High-Performance Computing Cluster Distribution area. A desirable enhancement to the Linux Kernel that could be added to the list in §6.3 would be incorporation of the *perfctr* patch that enables access to the hardware performance counters so that it is part of the mainstream Linux distribution and no longer requires a patch.

Another area with which interaction would be beneficial is the Resource Management area (Appendix B). Resource managers can help allocate and schedule the resources needed by the run-time tool communication infrastructure and by interactive run-time analysis tools. The run-time system can assist with automated run-time performance

instrumentation. Integration of tool invocation with the job scheduling system can improve ease of use of the tools and provide support for run-time interaction and real-time analysis. In the other direction, performance data produced by tools could be used to improve resource management decisions.

Web Sites and Selected References for More Information

Dyninst:
http://www.dyninst.org/

PAPI:
http://icl.cs.utk.edu/papi/

MRNet and TDP:

http://www.paradyn.org/

B.P. Miller, A. Cortes, M. Senar, M. Livny, "The Tool Daemon Protocol", SC'03, Phoenix, AZ, November 2003.

P.C. Roth, D.C. Arnold and B.P. Miller, "A Software-Based Multicast/Reduction Network for Scalable Tools", SC'03, Phoenix, AZ, November 2003.

SvPablo:
http://www-pablo.cs.uiuc.edu/

TAU:
http://www.cs.uoregon.edu/research/paracomp/tau/tautools/

MetaSim:
http://www.sdsc.edu/PMaC/