SLAC-PUB-14198

Collaborative Visualization for Large-Scale Accelerator Electromagnetic Modeling

Principal Investigator Dr. William J. Schroeder

Kitware, Inc. 28 Corporate Drive Clifton Park, NY 12065
http://www.kitware.com
518-371-3971

Work supported in part by US Department of Energy contract DE-AC02-76SF00515.

SLAC National Accelerator Laboratory, Menlo Park, CA 94025

Collaborative Visualization for Large-Scale Accelerator Electromagnetic Modeling
CRADA SLAC-331
Greg Schussman

This document contains the information requested for Project P-331, "Collaborative Visualization for Large-Scale Accelerator Electromagnetic Modeling", from the CRADA SLAC-331 agreement.

To be more specific, SLAC helped identify key features needed for synchronous and asynchronous collaborative accelerator environments. Readers were implemented allowing ParaView to read (in serial and in parallel) SLAC specific mesh files, field files, and particle files. Figure 1 shows SLAC mesh, field, and particle data read with these readers and rendered in ParaView. Key accelerator visualization procedures were streamlined into buttons on a SLAC toolbar for ParaView. These include visualization pipeline construction and adjustment, field selection, automated pseudo-color related scaling, mesh rendering styles, and line plots. SLAC participated in testing and debugging.

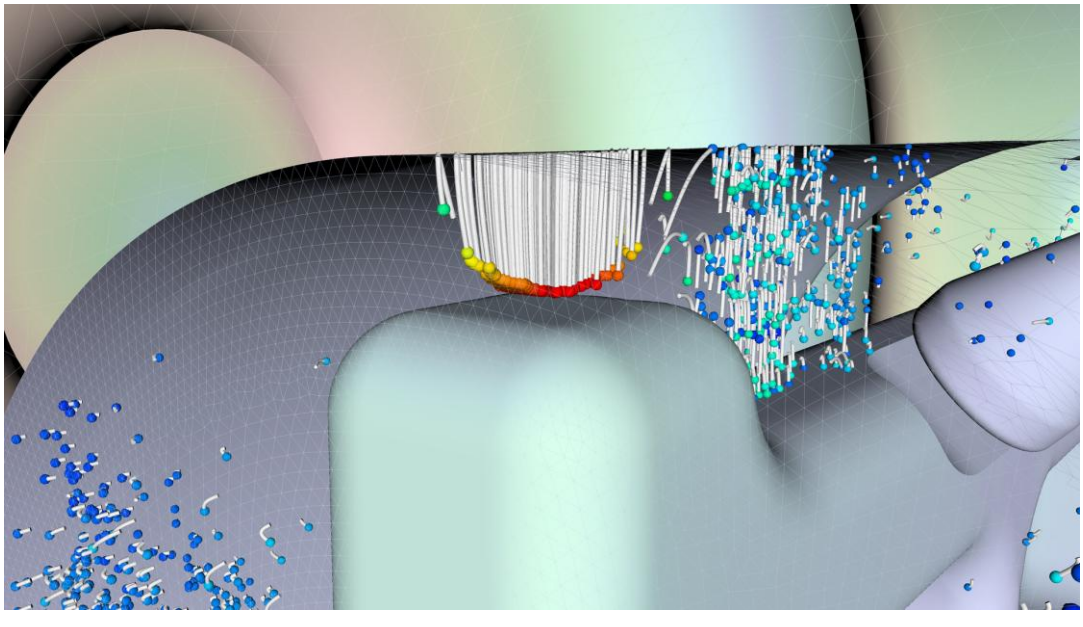


Figure 1: This image shows part of a multipacting simulation in the coupler region of an accelerator structure. The mesh file, field file, and particle files (all in the SLAC format) are read by ParaView using readers developed for Phase I of this SBIR. Particles are colored by momentum. Particle trails are shown in white. Electric field magnitude is indicated by pastel colors. This is one frame of an animation rendered in ParaView.

Phase I: Demonstration of Technical Feasibility

In the Phase I SBIR we proposed a ParaView-based solution to provide an environment for individuals to actively collaborate in the visualization process. The technical objectives of Phase I were:

- to determine the set of features required for an effect collaborative system;
- to implement a two-person collaborative prototype; and
- to implement key collaborative features such as control locking and annotation.

Accordingly, we implemented a ParaView-based collaboration prototype with support for collaborating with up to four simultaneous clients. We also implemented collaborative features such as control locking, chatting, annotation etc. Due to in part of the flexibility provided by the ParaView framework and the design features implemented in the prototype, we were able to support collaboration with multiple views, instead of a simple give as initially proposed in Phase I.

In this section we will summarize the results we obtained during the Phase I project. ParaView is complex, scalable, client-server application framework built on top of the VTK visualization engine. During the implementation of the Phase I prototype, we realized that the ParaView framework naturally supports collaboration technology; hence we were able to go beyond the proposed Phase I prototype in several ways. For example, we were able to support for multiple views, enable server-as well as client-side rendering, and manage up to four heterogeneous clients. The success we achieved with Phase I clearly demonstrated the technical feasibility of the ParaView based collaborative framework we are proposing in the Phase II effort.

We also investigated using the web browser as one of the means of participating in a collaborative session. This would enable non-visualization experts to participate in the collaboration process without being intimidated by a complex application such as ParaView. Hence we also developed a prototype web visualization applet that makes it possible for interactive visualization over the web.

Collaborative Visualization with ParaView

In the Phase I proposal, we proposed the development of a prototype focusing on the base capabilities in preparation for a full implementation in Phase II. The goal of this prototype was to demonstrate the feasibility of a collaboration using ParaView as well as define the whole user experience and the set of features one would like in a collaborative visualization tool.

ParaView is client-server based architecture for parallel visualization. The user connects to a server, typically a remote cluster with high compute power, using the ParaView client. Once connected to the server, the user then controls the visualization with the client while all the data processing, and optionally rendering, is done on the server side on the cluster. In the Phase I effort we extended this paradigm for a collaborative setup. All users that need to collaborate simply connect to the same server. The server instead of simply being driven by a single client serves multiple clients. Multiple clients bring in a new set of complications: which client has the control; how to communicate messages set from one client to the other; managing potentially duplicate data processing pipelines in each client, and so on. These and other issues are covered

in detail in the following subsections.

Leader and Participants

When multiple users are working on the same task, there must be a control resolution mechanism to ensure that only one user is modifying the visualization at any given time. This is achieved by categorizing the connected users into two types: one and only one Leader and several (if any) Participants. The Leader is the user who has the control over the visualization session. The Leader is the participant who drives the visualization; for example creating readers to read data, applying filters to process the data and controlling how the data is visualized. In other words, the Leader is the one who has the access to the full functionality of a standard ParaView client. The Participants, on the other hand, are the observers for the actions of the Leader. They cannot change the state of the visualization; they can merely observe what the leader sets up. However they have access to the introspection capabilities of the ParaView client i.e. they can open a panel showing the information about all the datasets being processed, or inspect the visualization pipeline that the Leader is controlling. There can only be one Leader at any given time, while there can be zero or more Participants.

In the Phase I prototype, the first client that connects to the server is assigned the Leader role by default. We also implemented a control locking and transfer mechanism in the prototype allowing the leadership to be fluid i.e. it can be passed around among the participating users. The next subsection describes how to pass the leadership using the Collaboration Manager.

A collaborative session does not have to wait for all participants to connect. A participant can join in on an existing session. In that case, we ensure that the newly connected participant is in the same state as the existing clients. Similarly, participants can leave in the middle of a session. There must always be one Leader in the session. If a Leader leaves a session, then a participant is randomly chosen as the new Leader. The server exits when the last client disconnects effectively terminating the collaborative session.

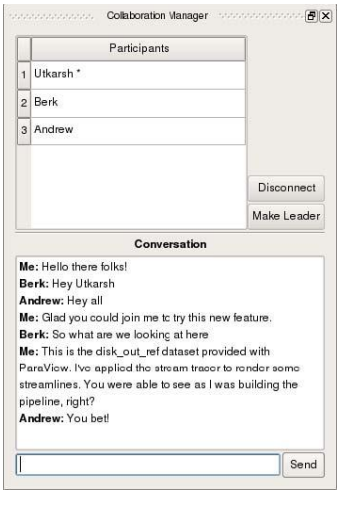
Visualization

The clients begin participating in the visualization process as soon as they connect to the server. Based on the leadership permissions described earlier, the leader is the only client that can control the visualization pipeline. The Leader has access to all the features provided by ParaView which includes opening files to read data, apply filters to process the data, using 3D widgets to change filter parameters, put annotation text for labeling, create animations etc. As the Leader goes about doing these tasks, all the other participating clients are updated simultaneously to reflect the resulting changes. For example when the Leader opens a file, all clients see that a file has been opened and a reader has been created. They can even inspect the values for the parameters the leader has set for the reader. In other words, all the participating clients behave as if the action was done locally.

The Leader can create views to show the data. As soon as these views are created, the other participants also reflect the action. When the data is rendered in the view, all the participants see the visualization as well. In the prototype we synchronized the viewpoint with the Leader, hence all the participants see the data exactly as the leader is seeing it. In the Phase I proposal, we had

proposed that we'll only be able to support single view for collaboration, even though ParaView supports multi-view configurations. However, by using the abstraction provided by the ParaView framework to our advantage and other architectural enhancements, we were able to support multi-view configurations including views such as x-y line plot view, bar chart view and spreadsheet view.

At any point in the visualization process the leadership can be transferred to any other participant using the Collaboration Manager panel. This makes it possible for the collaborators to actively participate in the visualization process. The prototype Collaboration Manager panel is shown in the figure to the right.



Collaboration Manager panel.

Rendering

The final stage of any visualization pipeline is generally rendering: mapping the data to the display. ParaView supports two rendering modes:

- Server-side rendering where the rendering is performed on the server and the images are shipped to and then displayed on the client. This mode has the advantage that size of the data delivered to the client is independent of the data being rendered. It depends only on the resolution of the rendered image. This is advantageous in large data visualizations where the rendered geometry can be huge. Also, since the server can be run in parallel on a cluster, it can employ parallel rendering techniques to improve rendering performance by distributing the effort. However since the server does the rendering, the client needs to fetch new images from the server on every interaction. This can result in jittery interaction over low bandwidth connections. Note that to alleviate interaction issues, ParaView provides a means to subsample the images delivered during interaction.
- Client-side rendering where the geometry to be rendered is shipped to the client and then the client renders the geometry locally. Since the client receives the entire geometry, it does not require the server unless the geometry changes. Thus, all rendering resulting from interactions with the camera can be handled locally. Also, the geometry size must be small enough to fit on the client.

Both these modes have benefits in different configurations: remote-rendering is preferred for large geometry setups, while client-side rendering is used when connections are slow and the geometry is small enough to fit on the client.

In the Phase I proposal, we indicated that we would implement remote-rendering support alone. However, with the help of several improvements to the way the data/image delivery components work, we are now able to support both local as well as remote rendering on a per client basis; i.e. each user can choose for itself whether to use remote-rendering or local-rendering based on parameters such as connection speed, local rendering capabilities, and so on. This is a huge advantage since it breaks the dependency of the client on the rendering process. Since each client can control the rendering process independently, each client can render at a resolution optimal for its display. Thus, clients can support varying screen resolutions and data sizes and can still work together without any one having to sacrifice on visualization quality. Hence, we are proposing support for heterogeneous clients including tiled displays for Phase II.

Implementation Details

This section covers the design details of the Phase I prototype. Before we delve into the details, we give a brief summary of the ParaView application framework. The crux of this framework is the ParaView ServerManager. The server manager is an abstraction layer that hides the complexities of client-server communication from the application layer, proving a unified façade irrespective of the underlying configuration.

ParaView ServerManager

ParaView is a parallel visualization application. It is designed to do all the data processing and/or rendering in parallel with several processors running over a cluster. We use MPI (Message Passing Interface) for communication between these processes. ParaView can also be used in

client-server configuration where the data processing is done on the server that may be running in parallel, while the client serves as the driver as well as the viewer for the visualization results. As briefly described previously, there are additional configuration options that control if the rendering must be done in parallel on the server-side or deliver the geometry to the client and render on the client side. The former is used for large geometry setups while the latter provides better interactive frame rates when the geometry sizes are small enough for the single client to handle.

To isolate the application layer from the intricacies of running is parallel and in client-server configurations, an abstraction layer called the ServerManager was created. The ServerManager provides proxies for every filter, source or mapper etc. created for processing/rendering the data on the server side (and sometimes on the client side as well). The application always uses the API provided by proxies to create pipelines and change parameter values. The proxies ensure that the based on the configuration those operations are sent to the right process to affect the actual source/filter objects. The ServerManager is an xml-configurable, xml-serializable layer. That makes it possible to provide a plethora of interesting features such as plug-ins, undo-redo, and state save-restore with ease.

With most of the client-server logic encapsulated in the ServerManager abstraction layer, the GUI layer can be thought of as a mere observer for the changes to the ServerManager state while providing mechanisms for affecting the same via panels, menus etc. This has made it possible to provide different clients for Paraview; e.g. the standard ParaView application Qt GUI, or a python client, while still reusing the core.

Implementation

The ServerManager is the abstraction layer that encapsulates the client-server communication. To create a filter on the server side, the application creates a proxy for the filter on the client side. This results in creation on the actual filter on the server side and setting up an association between the server-side filter and the proxy on the client. The proxy provides properties that are used to change the parameters on the filter on the server side. As mentioned earlier, these proxies and properties are XML serializable. Hence, it is possible to restore the state for a pipeline by recreating all the proxies and restoring their property values. This principal forms the basis of our design.

As the Leader sets up the visualization pipeline creating new proxies and changing their property values, we serialize these changes as XML and ship them to all other connected participants. When a participant joins a session already in progress, we simply ship the state for the entire visualization pipeline (instead of just the changes) to the newly connected participant. Once the participants receive the XML, they load that XML to create new proxies or change properties on existing proxies etc. These XML changes can include complex changes to the camera, or instantiating new view windows.

The ParaView GUI layer is designed using a model-view paradigm, where the ServerManager serves as the model for the visualization pipeline while the GUI acts as the view. As and when new proxies are created in the ServerManager or when their property values are changed, the GUI updates itself to reflect the changes. Hence when the XMLs are loaded on the participants, their GUIs reflect these changes as if it were done locally. Thus the participating

clients remain in sync with the Leader.

The communication of XML packets from the Leader to the Participants happens via the server process. Every atomic change on the Leader is sent to the server that then broadcasts it to all other connected clients. For the prototype we decided to route all the inter-client communication via the server. This has the advantage that the clients don't have to be aware of one another. However this also implies that the server has to do the additional work of providing a communication channel. For Phase II, we would like to experiment with support for direct client-to-client communication using peer-topeer technologies widely used by applications such Skype™and Google Talk™.

Sharing Data Pipelines

As mentioned earlier, a proxy on the client represents a filter (or a processing unit) on the server side. When a new proxy is created a new server-side filter is also instantiated and there's logic in the ServerManager to keep the two associated with each other.

When collaborating with multiple clients, all clients are connected to same backend data processing and rendering server. Since all pipeline objects are on the server side, we can very easily share these objects with all connected participants. Thus all participants will have their client-side proxies referring to the same server side pipeline objects. Thus not only keeps the server side memory overhead for each client minimal but also gains from shared data processing for all participants.

At the same time, we can still support creating of non-shared pipeline objects i.e. participants can create proxies (with associated server-side pipeline objects) that are not accessible to others. This will enable us to provide support for local exploration for Phase II, as described later in the Phase II project description.

Web Visualization: Collaborating over the Internet

In recent years the web has been gaining popularity as a medium for communicating information and collaborating. Internet applications are becoming more popular and new ones are developed for as diverse domains a financial bookkeeping to photo editing to gaming. The visualization community already uses the Internet extensively for sharing data as well as information using Web 2.0 based frameworks such as MediaWiki. A natural evolution is to support visualization collaborations via the standard web browser. Hence, for Phase I, we also investigated approaches for supporting a web browser as a participant in a collaborative visualization session.

The core of the support for collaboration in ParaView is implemented in the ServerManager layer. That makes is possible for heterogeneous clients to participate in the collaboration session, as long as all the clients are based on top of the ServerManager layer.

We investigated a couple of different approaches. The goal was to provide a web component that web site developers could plug into their website to add support for interactive visualization. The web-service based solution also made it possible to use the browser as a scripting environment for ParaView, enabling website developers to create and configure visualization

pipelines. The Flash™ based solution focused on improving interactivity by using server side technologies for better streaming for rendered images to the client. For Phase II, we plan to implement a solution encompassing both these features, allowing configurability without sacrificing interactive performance. In the following subsections we discuss the different approaches for web visualization.

SOAP-based Web Service with Javascript Client

A web service can be thought of as a server-side component that provides a defined service to the connecting clients –in our case, the service is data visualization. SOAP is a protocol specification for exchanging data for web services. Using a standard protocol for the web service makes it possible for different clients to connect and use the web service. Also several client as well server side libraries are currently available that make it relatively easy to develop client/server components.

In our implementation, we used Zolera as the server-side SOAP infrastructure for developing and deploying our web service. One determining factor for using Zolera was the fact that it is Python based. Since ParaView already has a Python scripting interface, which is nothing but a Python-based client over the ServerManager layer, it was relatively easy to expose the functionality provided by ParaView's scripting API as a web-service.

On the client side code, we started with a simply Javascript-based browser component to show rendered images. Since SOAP is an XML-based format, the messages can be large and cumbersome to process on the browser. Since web browsers are optimized to parse JavaScript, a format called JSON (JavaScript Object Notation) has been gaining popularity with AJAX-based websites.

Hence, we wrote a simple Python-cgi script that acted as a JSON-to-SOAP bridge. This bridge accepts JSON messages from the web browser, translates them to corresponding SOAP requests and then forwards them to the ParaView Web service. It then translates the SOAP response to either a JSON reply or an image fetch response (when rendering images).

System Architecture for Web Visualization using a SOAP-based Web Service.

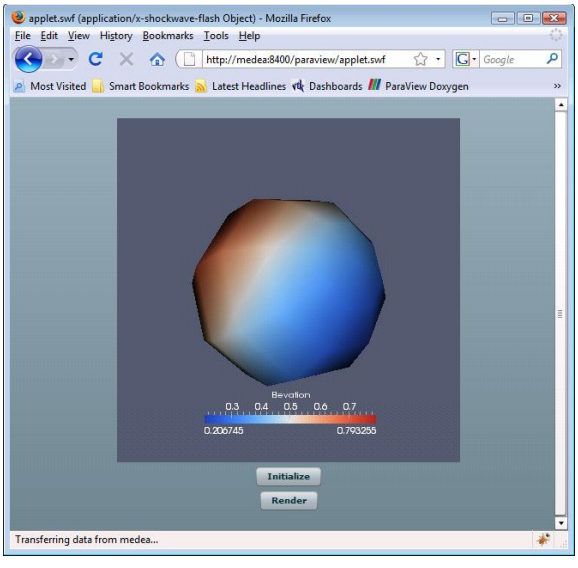
Since ParaView's Python scripting API can be exposed via the web service, we wrote a module that makes is possible to write JavaScript scripts, similar to the Python scripts for creating the visualization pipeline. The following is an example script for visualizing a dataset:

```
// Create the reader. var exodusReader =
paraview.ExodusIIReader({FileName="can.ex2"}); paraview.Show() // Show the rendered
image. paraview.RenderImage($("renderWindow"));

// Apply a Shrink filter to the reader. var shink = paraview.Shrink({Input=exodusReader})
paraview.Show() // Show the rendered image.
paraview.RenderImage($("renderWindow"));
```

We implemented a JavaScript Ajax-based library that provides the API demonstrated above. This makes it possible to use the web browser as simply another scripting environment for the ParaView engine. Also, this scripting API is very similar to the Python scripting API that ParaView already supports.

Although the SOAP-based web service provides for a powerful web visualization solution, we discovered that the parsing of SOAP XML resulted in considerable overhead. That adversely affected the frame rates and we were not able to realize frame rates better than 5 fps for simple visualizations. Also since all communication was over HTTP via an Apache web server, there were no persistent connections over which the communication was taking place. The HTTP connect-request-response protocol also added to the overhead of each communication message sent by the client.



SOAP Web Service with Adobe® Flash™ Client

One of the critical characteristics of the web client is its interactive rendering speed. Since the frame rates we achieved with the pure JavaScript/HTML based solution described earlier was less than 5 fps, we decided to use Adobe Flash™ technology for developing the client side plug-in. Another advantage of using Flash is the encapsulation of the client side code into a single applet which can be plugged into any website.

Adobe Flash™ provides remoting components that make it easy to communicate with web services based on standard protocols such as SOAP. We implemented a flash applet to act as a SOAP client that directly connects to the SOAP web service (see figure to the right). Although this communication bypassed the Apache web server, it was still implemented over HTTP (since Zolera supports servicing HTTP requests as well). Using Flash for the browser plug-in made it easy to support interaction with the visualization. However, we still weren't able to get frame rates better than 6-7 fps. Since rendering the image was taking under 0.05 seconds, we concluded that the non-persistent HTTP connections and the overhead due to SOAP resulted in low frame rates. Since we believe that for any commercially viable solution frame rates of 15 fps or better are absolutely critical, we decided to try other alternatives.

Adobe® BlazeDS™ Web Application with Adobe Flash Client

Adobe® recently released BlazeDS™ which is a server-based remoting and web messaging technology that enables Flash™ clients to connect to various backends for real-time binary data communication. It also provides JavaScript/AJAX library to enable JavaScript based clients to communicate with the BlazeDS server-side components. BlazeDS is an attractive architecture for our web visualization prototype. It provides persistent binary communication channels with the server thus supporting streaming of data efficiently. It's easy to deploy a BlazeDS based application on any Java-based Web Server such as Tomcat. It supports real time message and streams and yet, is over HTTP thus not requiring any additional ports to be opened on the server side. Since BlaseDS supports raw binary data communication, it avoids the need to use base64 or any other encoding to convert binary data to strings. Finally, communication with BlazeDS web application using Adobe Flash clients is fairly straight forward since the Flash remoting API provides components that simplify setting up communication streams, calling remote methods, and so on.

The results we obtained using BlazeDS™ are very promising. We were able to achieve a frame rate up to 12 fps. We intend to explore this option further in Phase II.

Challenges and Limitations

In Phase I we set out to develop a prototype based on the ParaView framework for collaborative visualization. We were successful in demonstrating that ParaView's client-server paradigm can be extended to support a collaborative environment where multiple users collaborate in the visualization processes. This section summarizes some of the major challenges we faced and highlights some of the limitations of the prototype. We will address these limitations in the Phase II implementation.

- One of the first issues we face in deploying the proposed prototype in organizations is with firewalls. Organizations typically have firewalls blocking incoming connections. This makes it difficult for collaborator to connect to the server if the server is behind a firewall. As described later, we plan to explore technologies used by VOIP applications such as NAT traversal to overcome issues related to firewalls eliminating any need for any firewall holes for collaboration.
- As described in the implementation, in our prototype the server process acts a communication hub for all communications between the collaborators. This implies that the server has to spend time relaying the messages when it could be processing data or rendering it. There are a couple of possible solutions for this: we can use a multithreaded server thus delegating the relaying to a separate thread; or we can use the peer-to-peer technologies similar to those employed by applications such as Google Talk ™ and Skype ™ to directly establish communication channels between the clients.
- In the prototype the rendering can be done on the server or the client. When rendering on the server, the server renders separately for each connected client. This certainly has the advantage of being able to provide optimal resolution images for all the connected clients; however, it may overload the server affecting response times during interaction, since many clients could be requesting renders at the same time. For Phase II, we propose to investigate a solution based on reusing rendered images where ever possible to avoid repeated renders, thus allowing us to support more number of collaborating participants.
- A limitation of the prototype is that it does not support configurations where one of the connected clients could be running a tiled-display. Supporting heterogeneous participants is one of the distinguishing factors of our design and hence we plan to implement it for Phase II.
- The web visualization component developed in Phase I does not support connecting to a

collaboration server. It also requires that the visualization server is same machine as the web-server. This is huge restriction in real world scenarios since generally the visualization server will be on a high performance cluster. We will overcome these limitations as a part of Phase II.

- Another important issue is authentication and encryption. The prototype is a free-for-all setup. Participants can join on to a collaborative session by simply connecting to the same server. We need authentication to ensure that only authorized participants join in and encryption to ensure that the data being communicated is secure.

These and other issues will be addressed in the Phase II effort. In summary, we were encouraged by the progress that we made in the Phase I effort, going well beyond our initial objectives. Based on these initial results, we are confident that we can develop a solid implementation in the Phase II project.

FINAL REPORT

for

Project entitled:

Collaborative Visualization for Large-Scale Accelerator Electromagnetic Modeling

Partner:

Kitware, Inc.

DOE Laboratory:

SLAC National Accelerator Laboratory

This work was supported under its U.S. Department of Energy Contract,
No. DE-AC03-76SF00515

SLAC Project Manager:

Greg Schussman

This final report is being submitted to meet the requirements in the CRADA agreement

CRADA SLAC-	331

As stated in the following paragraph.

ARTICLE XI: REPORTS AND ABSTRACTS

The Parties agree to produce the following deliverables: an initial abstract suitable for public release; and a final report to include a list of Subject Inventions. It is understood that the Contractor has the responsibility to provide this information at the time of its completion to the Contracting Officer and the DOE Office of Scientific and Technical Information.

Date submitted:	2010 JUN 07
Submitted by:	James E. Simpson