

# A Framework for Reuse and Parallelization of Large-Scale Scientific Simulation Code

Manolo E. Sherrill<sup>a</sup>, Roberto C. Mancini<sup>a</sup>, Frederick C. Harris Jr.<sup>b</sup>, Sergiu M. Dascalu<sup>b</sup>

<sup>a</sup>Department of Physics

<sup>b</sup>Department of Computer Science and Engineering  
University of Nevada, Reno NV 89557

## Abstract

*Software design in the scientific community often fails due to the nature and life span of the code being developed. In this work a software framework is proposed for large scale scientific computations. This architecture is applied to simulations of laser ablation, in particular Li and Ag. This architecture also allows expansion to parallel computation without any additional work.*

**Keywords:** laser ablation, software framework

## 1 Introduction

Several difficulties affect code development in scientific programming. The often short lived nature and highly changing aspect of most scientific programs produce a dilemma for the developer. Small programs are crafted in a temporary manner to test algorithms and methods to solve specific computational problems. In cases where a code's "lifetime" exceeds expectations, it becomes very difficult to incorporate it into larger simulation programs due to variable name conflicts, poor organization, non-robust implementation and a lack of consistent style. In regards to a pre-existing simulation program, additional complications arise when new projects are undertaken that require large modifications to the execution topology. It is impractical in most cases to place various options into a program in anticipation of future project requirements. Older simulation programs have the added difficulty of being overburdened with obsolete or unnecessary functionality for the project at hand and thereby complicating the adaptation procedure. Unfor-

tunately, the removal of un-needed routines leads to a high probability of introducing errors into the program.

Our work has focused on scientific simulations of laser ablation. This field of physics fits the description just provided with pieces of code that have been written once and used, and modified over and over again, for years. Like any scientific field the simulations have to change as the real world knowledge increases. This knowledge will increase as experimentalists look to confirm or study data presented by the simulations. Therefore, the code has the potential to be used over and over again and be modified many times at unknown frequencies. We found in our simulations that the standard sequential development models did not fit or work effectively in this field and we developed a new framework that we propose.

This paper, in its remaining parts, is organized as follows: Section 2 presents a background into the laser ablation simulations that precipitated the architecture. Section 3 describes the framework design that is the foundation for this work. Section 4 gives some results. Section 5 follows with conclusions and future work.

## 2 Laser Ablation

Laser ablation refers to the process of ablating material from a solid or liquid target, with a low intensity laser ranging from  $1 \times 10^7$  W/cm<sup>2</sup> to  $1 \times 10^{10}$  W/cm<sup>2</sup>. Typically, a pulsed laser is used to irradiate the target. It deposits the bulk of its energy in the skin depth region of the tar-

get where this volume of material is heated and then undergoes melting, evaporation, and possibly plasma formation. The material in the gaseous state then forms a plume that expands away from the target's surface with normal to the surface velocities of a few  $10 \mu\text{m}/\text{nsec}$ . When a series of ablation events are performed, where the duration of the irradiation of the target is allowed to stay constant (that is the full width at half maximum (FWHM) of the temporal pulse shape is fixed) while the fluence is allowed to increase, a transition between evaporative plume to plasma plume formation can be observed [10].

Laser ablation is commonly used in both experimental physics as ion sources, and in industry for the generation of thin films. In fact, laser ablation has proven to be the most consistent method of producing high quality thin films, in particular, for stoichiometrically complex material. This technique has been used in the manufacturing of electronic and optical films, super-conductors, ferroelectrics, piezoelectric and photoelectric materials as well as tribological coatings such as diamond like thin films [2]. In the last 4 years, more exotic systems have entered this application arena. Pico- and femto-second lasers have found their place in the thin film synthesis and annealing. A further exotic ablation process is matrix-assisted laser desorption ionization (MALDI), used to place large biomolecules, such as proteins, in a free environment for use in mass-spectroscopy and other studies [9].

From the early to mid 1960's, after the availability of the first ruby lasers, a substantial effort toward the understanding of laser matter interaction from both the theoretical and the experimental perspective was under way. Solid, liquid, and gas target interactions were all investigated. These investigations lead researchers to think of possible applications; and by 1965, the laser was successfully shown to be a useful tool in producing thin films on a substrate [13]. For the generation of thin films, laser driven ablation waned.

Laser ablation acquired new interest during the late 1980's for a series of work done on the synthesis, in particular, of high quality stoichiometrically complex high temperature superconductor ( $\text{YBa}_2\text{Cu}_3\text{O}_{7-x}$ ) [4] through the use of pulsed laser deposition. Pulse laser deposition

(PLD) produced a greater congruent ablation than other deposition techniques. The shorter pulse durations allowed for the thermalization of a shallower volume of target material. This preserves the stoichiometric properties during the transfer of the material to the substrate where the thin film is grown. With the accessibility of higher frequency lasers (Nd: YAG  $1.06 \mu\text{m}$ ), the target volume accessed directly by the laser energy was also reduced. This added to a greater congruent ablation, as well as a reduced effect of subsurface heating - the main cause of splashing: the ejection of molten globules from the surface of the target. By 1992, Saenger, K. [2] reported over 180 thin films synthesized with PLD. These materials included metals, inorganic and organic compounds as well as polymer films; and in the last few years, PLD has been used in the development of nano-materials such as the synthesis of carbon nanotubes.

Though there has been a large body of experimental work dedicated to the characterization of ablation plumes produced under various experimental parameters and targets, little attention had been paid toward a fundamental understanding of simple systems. This is primarily the outcome of the ablation communities' desire to have information on specific systems for the synthesis of particular materials. It is also recognized that the complexity of even the simplest systems can elude theoretical characterization; due in part, to the myriad of possible plume constituents such as atoms, atomic and molecular ions, clusters, and micron size particles, whose abundances may change with a small change in laser fluence.

Work that has attempted to describe ablation physics, in general, has resided either in the detailed modeling of laser target interactions [5] or the modeling of the expanded plume through gas dynamic simulations [16] or hybrid models [17]. Interestingly, detailed study of the plasma in the region between the target and a few millimeters from the target has not been undertaken [7]; although it is this region of the plasma that defines the level populations and the ionization abundances of the plasma far away and later in time (in the absence of background gases).

Our work focuses on the modeling and analysis of a laser ablation plume for the region from

tens of microns to a few millimeters away from the target surface and early in time (20-100 nsec after the end of the laser pulse). This work attempts to provide detailed quantitative information of laser ablation in this spatial and temporal regime for a modest stoichiometrically complex target.

### 3 Framework Development

During the development of the multi-element and multi-spatial zone spectroscopic model, timings recorded for several execution trials indicated that our initial program written in a sequential form required an unacceptable amount of time to execute. Due to this fact a parallel implementation of the simulation code was pursued. For various technical reasons the program became too cumbersome to confidently perform modifications to include new physical effects even though good programming practices were employed. This condition is unfortunately not uncommon when dealing with complex simulation programs [3]. The solution would lead in a new direction of software development for this project.

This section discusses the specific difficulties in dealing with complex simulation codes and draws from work done by the system developers community to describe a new methodology for constructing and implementing research simulation programs. As an example, this method is applied in the new spectroscopic code developed for this work. However, the underlining ideas are general and can be applied to other research codes as well.

#### 3.1 Paradigm

For several years operating system developers have dealt with similar problems as those faced by computational physicists in regard to engineering software. From their work two predominant paradigms for operating system architectures or kernel architectures have emerged. The traditional monolithic kernels (typical of most UNIX operating systems and similar in form to most physics codes) are characterized as a collection of procedures typically compiled separately and linked into a single large executable code.

They are often implemented in a small number of layers. Protection, however, from the corruption of data (encapsulation) by other layers or procedures is non-existent. Interestingly, monolithic operating systems have noted examples where the complexity of the program grew to such an extent that the modifications needed to remove bugs led to the introduction of new bugs and to the eventual abandonment of the operating system [15].

The other kernel architecture is exemplified by the design of the microkernel. Here only a very small set of functions are included in the kernel. The remaining functionality needed by the operating system is included (i.e. memory management, file system services, etc.) as separate modules and are run as separate processes as needed. Interprocess communication between modules occurs through message passing. Though modules can communicate with each other, often for many operations the microkernel acts predominately as a centralized point of connection and communication[6].

To clarify the meaning of a modular implementation we must describe the qualities of a module [8]. A module is allowed to depend only on the interfaces of other modules and not on their implementations. This quality immediately precludes the use of global variables between modules. Modules are designed to encompass a large element of functionality such as a memory subsystem. A purely modular implementation allows modules to interconnect freely with each other. This is in contrast to a layered model where connectivity is limited to elements "above" an "below" a given element [6].

Though operating systems based on a microkernel design are usually slower than the monolithic ones due to increased overhead of message passing they do possess many advantages that may be enlisted in dealing with software engineering problems found in simulation codes. The microkernel structure forces system developers to employ functional components in a modularized manner since they are ran as separate processes. This will be one of the most important qualities exploited for this new method. Communication between processes only occurs through well-defined and clean software interfaces - requiring all transferring variables to be

listed at the interface. This makes it easy to maintain, develop and replace modules without affecting the rest of the system. Moreover, message passing facilitates the creation of software where tasks can be distributed among several computers to lower overall execution times. One other positive feature of microkernel operating systems is their tendency to use random access memory (RAM) more efficiently than monolithic ones, since they have the capability to create or destroy processes (functionality) as the need arises. This is in contrast to the monolithic implementation where executable instructions and data of various functions remain in memory - even after they are needed - until execution is completed [1].

### 3.2 Implementation Overview

Though the low level message passing primitive used in the microkernel design is not appropriate for scientific applications, the technique provides motivation for exploring the use of parallel message passing (PMP) libraries as a means of implementing a microkernel like strategy. In using a PMP libraries, application modules like the microkernel case run as separate processes and in turn separate memory address spaces - each processes is controlled and protected by the operating system. Pathological modules that access memory outside of their specified spaces cannot corrupt variables in other modules. This quality thereby reduces the time needed for diagnosing problems. Furthermore, in using PMP libraries, the execution of processes is no longer limited to one machine. Inherently, two desirable conditions have been obtained - true modularity and parallel capability.

At this point the most drastic deviation from the microkernel design is made - the elimination of the central program from which modules are typically mounted (the microkernel itself). The removal of this hub like structure alleviated two technical difficulties: First was the need for adding variables and complexity to the central program simply to transport data from one module to another and secondly, in regard to a parallel implementation, to prevent a network bottleneck from occurring at the computer node that contained the central program. This

modification facilitated a more peer style implementation where modules are connected to each other like Tinker-Toys and the program topology resembles the natural interconnections of the subject being modeled.

With the removal of a centralized data transferring module went a convenient process control center - recall that the microkernel added and removed processes as needed to improve efficiency. To retain this feature a new hierarchical program structure was needed - an ordered multi-layer model was chosen where a process communicates only directly with members of its own layer or adjacent layers [11]. Processes are spawned by the next higher-order parental layer. The removal of a process is signaled typically by its parent, or more rarely, by a module within its own layer.

In this structure constraints on the interconnectivity of a given module have led to an implementation that has characteristics of both modular and layered schemes. Though the implementation is far from ideal in regards to either scheme an important benefit has been gained: the employment of layers prevents overly complex program topologies that may have otherwise occurred in a purely modular design. As in any layered model, signals and to a lesser extent data must be passed through to target modules residing deeper within the structure. Though this relaying of information from layer to layer may seem contrary to the original removal of the microkernel, it is noted that within each layer many modules are usually involved in the transporting of information and with modules deployed across many computer nodes the probability of a bottleneck is severely reduced.

As mentioned earlier PMP libraries are used to communicate between processes. Specifically, the Parallel Virtual Machine (PVM) libraries developed at Oak Ridge National Laboratories were used. To implement the layered structure of modules a new set of libraries referred to as the Workbench libraries was developed on top of the PVM libraries to assist in common tasks used in spawning and communicating between layer processes. The use of the Workbench libraries circumvents the main problem of using PMP libraries directly - often, coding becomes tedious and programs become too cumbersome,

in particular for a multi-layered program, when PMP libraries function calls are used directly in source code, thereby destroying the original intention for this development.

The Workbench library acts as a set of utilities built on the virtual machine presented by PVM much like UNIX utilities are built on top of the virtual machine presented by the UNIX kernel. These utilities transport data between modules without the requirement of specifying data type and in regards to arrays reduces significantly the number of function calls to initialize a transfer. Furthermore these libraries maintain data structures for accessing processes, tasks and processes locations (computer node). In addition, functionality has been added to send information directly to layers for file I/O, thereby reducing the amount of data that must be relayed through higher order layers.

## 4 Framework Instantiation

The techniques of the development scheme described in the previous section were applied to the spectroscopic model discussed in Section 2 (and in more detail in [12]), in particular, to the highest density case - occurring early in time and close to the target surface. Included in this model is the capability of calculating a gradient in the direction along the line-of-sight of the spectrometer. The existence of this gradient was discussed in Chapter 2 of [12], in regards to the experimentally observed self-reversal feature in the Li: 3d-2p lineshape. To accommodate a gradient, the theoretical plasma is divided into zones - each containing the same abundances of each species but each described by a unique temperature and atom number density.

For this plasma environment, optical depths are large, thereby requiring a separate calculation for the radiation transported through the different zones. The radiation from one zone does not, in this plasma, affect the population of another. From this assumption the atomic kinetics of each zone is left uncoupled and can be calculated independently. From these qualities a three layer model can effectively be constructed.

It should be re-emphasized here before going any further that the modular framework is

simply a framework. Except for the amount of data transferred between processes and for the topology of the implementation, the modules are independent from the physics codes embedded into them. Or in other words the modular framework constitutes a software network that the physics codes communicate through.

The Framework that has been developed is basically a set of library routines that allow us to write simple programs to handle all of the communication between legacy code modules. These routines allow us to separate the computation from the communication (since the legacy code does not know anything about communication). This separation allows the code developers to separate sequential physics routines and therefore parallelize on a coarse grained level.

This communication is illustrated in the figures of this paper via lines connecting triangles. The triangles and diamonds represent the communication interface that is provided by the Workshop libraries. The circles inside of the triangles represent the legacy simulation code sometimes in Fortran and sometimes in C. Triangles pointing to the right have the capability to spawn processes and have typically spawned the processes they communicate to the right with.

In our implementation, the lowest layer modules (Layer IV) contain a set of single element kinetic models (SEKM) represented by the circles that compute the populations of each element found in a particular zone of the plasma. The next higher layer of modules, the parental layer (Layer III) of the lowest layers, contains the self-consistency routines (SC) that check the multi-element kinetic calculation for completion (see Figure 1).

Once a self-consistent solution is obtained, the SC layer then signals the SEKM for emissivity and opacity data that is then relayed to the next higher layer - the radiation transport layer (Layer II). Here, data from each zone is accumulated and used to generate the synthetic spectra for one complete plasma. It is also here where experimental lineouts and the theoretically created spectra are compared (see Figure 2). Layers II-IV represent the synthetic spectral object (SSO).

The comparison of the experimental data to synthetic spectra is done in an automated man-

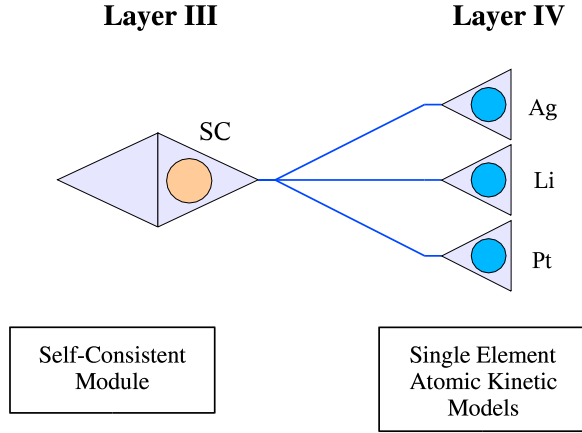


Figure 1: Single plasma zone multi-element atomic kinetic object [Layers III and IV] are shown.

ner by the use of a search engine. The search engine typically generates a large number of temperature and density profiles for which the physics model must produce a quantitative comparison between theoretical and experimental spectra. The temperature and density profile generated by the search engine is stored in a parallel queue (PQ). Initially, PQ spawns several synthetic spectral objects that load atomic and spectral data and then wait ready to generate synthetic spectra from the profiles dispensed from the PQ. The number of SSOs generated depends upon the computer architecture that the simulation is run on but can be as large as allowed by the number of CPUs, See Figure 3. The search engine, in search for the best synthetic to experimental spectral fit, generates thousands of profiles. Once the search is complete, PQ sends termination signals to each SSO. In each SSO a signal is relayed to the deepest layer where termination first begins. Termination continues to propagate up until all the SSOs are destroyed.

## 5 Conclusions and Future Work

In this paper we have presented the motivation for a change in software architecture for

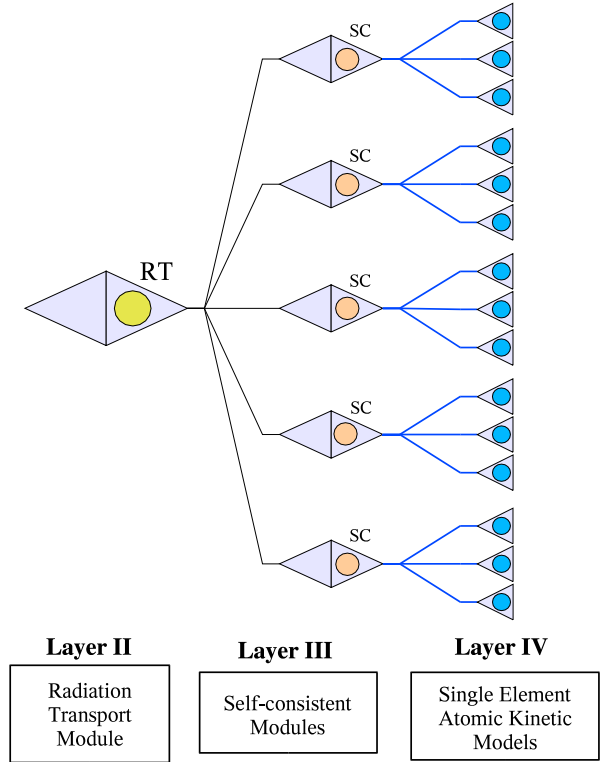


Figure 2: Synthetic spectral object involving five plasma zones [Layers II, III and IV] are shown.

maintenance and increased performance of simulation codes. This need resulted in the development of a new library, called Framework, that allows for protection and encapsulation of existing legacy code as well as allowing parallelization of that sequential code.

Framework is beneficial for simulations for a variety of reasons [14]. First when the experimentalists discover data that requires a modification to our code we do not have to recompile everything. Because the modules are wrapped into separate executables (and processes) we only have to recompile and relink that module when changing specific components of the model. Second, the physics components can be tested and modified individually before adding them to a larger simulation. Third, it allows us to change the topology quite easily and allows parallel processing. Fourth it helps keep the legacy simulation codes separate, thereby providing some pro-

tection from the use of code written by someone else.

In the future we are looking at applying this architecture to a variety of other physics codes that our research group uses. This will allow us to effectively re-use the original legacy code. The separation into separate processes will allow easier modification and maintenance of the code and will allow us to utilize the parallel architecture to increase the performance of the original code.

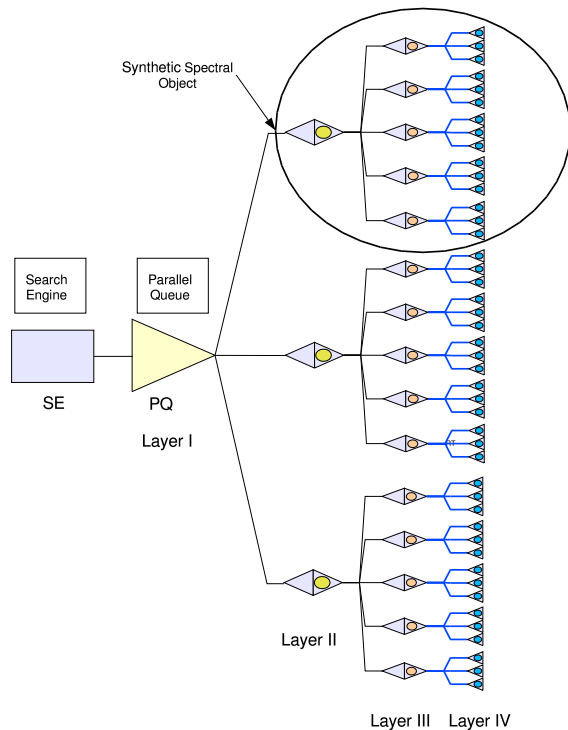


Figure 3: Diagram showing the communication of the parallel queue with three concurrent plasma spectral models (synthetic spectral objects). Layers I-IV are shown. The plasma executables remain in memory processing jobs until the parallel queue (PQ) is empty.

## References

[1] D. P. Bovet and M. Cesati. *Understanding the Linux Kernel*. O'Reilly, 2001.

[2] D. B. Chrisey and G. K. Hubler. *Pulsed Laser Deposition of Thin Films*. Wiley-Interscience, 1994.

[3] J. Cooling. *Software Engineering for Real-Time Systems*. Addison-Wesley, 2003.

[4] D. Dijkkamp and et al. *Appl. Phys. Lett.*, 51:619–621, 1987.

[5] P. Lorazo, L. J. Lewis, and M. Meunier. Simulation of picosecond pulsed laser ablation of silicon: The molecular-dynamics thermal-annealing model. *Proc. SPIE*, 4276:57–61, 2001.

[6] S. A. Maxwell. *Linux Core Kernel Commentary*. Coriolis, 2001.

[7] J. C. Miller and R. F. Haglund, editors. *Laser Ablation and Desorption*, pages 255–289. Academic Press, 1998.

[8] R. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 6<sup>th</sup> edition, 2004.

[9] A. A. Puretzky, Geohegan D. B., G. B. Hurst, and M. V. Buchanan. Imaging of vapor plumes produced by matrix assisted laser desorption: A plume sharpening effect. *Phys. Rev. Lett.*, 83:444–447, 1999.

[10] L. J. Radziemski and D. A. Cremers, editors. *Laser-Induced Plasmas and Applications*, pages 1–67. Marcel Dekker, 1989.

[11] J.A. Rolia and K.C. Sevcik. The method of layers. *IEEE Tran. on Soft. Engr.*, 21(8):689–700, 1995.

[12] Manolo E. Sherrill. *Spectroscopic Modeling and Characterization of a Laser-ablated Li-Ag Plasma Plume*. PhD thesis, University of Nevada, Reno, May 2003.

[13] H. M. Smith and A. F. Turner. *J. Appl. Opt.*, 4:147–148, 1965.

[14] I. Sommerville. *Software Engineering*. Addison-Wesley, 7<sup>th</sup> edition, 2004.

[15] A. S. Tanenbaum and A. S. Woodhull. *Operating Systems Design and Implementation*. Prentice Hall, 1997.

[16] R. F. Wood, K. R. Chen, J. N. Leboeuf, A. A. Puretzky, and D. B. Geohegan. Dynamics of plume propagation and splitting during pulsed-laser ablation. *Phys. Rev. Lett.*, 79(8):1571–1574, 1997.

[17] R. F. Wood, J. N. Leboeuf, K. R. Chen, Geohegan D. B., and A. A. Puretzky. Dynamics of plume propagation, splitting, and nanoparticle formation during pulsed-laser ablation. *Applied Surface Science.*, 127-129:151–158, 1998.