# AHAB
# — OR —
# The Hunt for Automated Shock Timing

by

Daniel J. Segal

A thesis submitted in partial fulfillment of
the requirements for the degree of

Bachelor of Science

(Engineering Physics)

at the

UNIVERSITY OF WISCONSIN–MADISON

2014

# Contents

Contents . . . . . . . . . . i

List of Figures . . . . . . . . . . ii

Abstract . . . . . . . . . . iii

1 **Introducing Shock Ignition** . . . . . . . . . . 1

1.1 Confining a Plasma using its own Inertia . . . . . . . . . . 1

1.2 Achieving Ignition with Aligned Shock Fronts . . . . . . . . . . 2

2 **Constructing a Laser Profile** . . . . . . . . . . 5

2.1 Imposing a Picket Pulse . . . . . . . . . . 5

2.2 Setting up Pedestal Pulses . . . . . . . . . . 6

2.3 Optimizing the Compression and Igniter Pulses . . . . . . . . . . 6

3 **Modeling Shock Ignition** . . . . . . . . . . 7

3.1 Using Bucky: A One-Dimensional Radiation-Hydrodynamics Code . . 7

3.2 Asserting Appropriate Approximations . . . . . . . . . . 8

4 **Running Multiple Instances of a Program** . . . . . . . . . . 9

4.1 Conducting Parameter Sweeps with AHAB . . . . . . . . . . 10

4.2 Utilizing HTCondor and a Network of Computers . . . . . . . . . . 13

5 **Conclusion** . . . . . . . . . . 14

A **The Radius, Time, and Temperature Plot** . . . . . . . . . . 16

B **Bucky Start Mode Capabilities** . . . . . . . . . . 21

**References** . . . . . . . . . . 24

# List of Figures

1.1 Inertial Confinement Fusion . . . . . 1
1.2 The National Ignition Facility . . . . . 2
1.3 Laser Power versus Time Profiles . . . . . 3
1.4 Gain Curve for Shock Ignition . . . . . 4

2.1 A Laser Power versus Time Profile . . . . . 5
2.2 The Picket Pulse . . . . . 6

4.1 A Scalar Parameter Sweep . . . . . 10
4.2 A Vector Parameter Sweep . . . . . 11
4.3 An AHAB Input File . . . . . 12
4.4 HTCondor . . . . . 13

5.1 Shock Convergence . . . . . 14
5.2 Automatically Timing Pedestal Pulses . . . . . 15

A.1 The RTT Plotting Function . . . . . 16
A.2 Regions Mode vs. Lines Mode . . . . . 17
A.3 Manual Zoom Input Box . . . . . 18
A.4 Plotting Multiple Files . . . . . 20
A.5 The Points Layer . . . . . 20

B.1 A Bucky Input File . . . . . 21

## Abstract

Shock Ignition is a form of ignition for Inertial Confinement Fusion, which promises gains that are nearly an order of magnitude larger than NIF baseline values. Although a relatively new scheme, Shock Ignition has received a considerable amount of research attention due to its ability to be applied to currently implemented technology.

A major component of Shock Ignition is computational modeling. One fundamental problem involved in it is the timing of laser-induced shocks, so that they coalesce at the same radius of their target. AHAB – or Automated High-throughput computing Achieved with Batch processing – was designed to help with this timing process. It has since grown to be capable of conducting parameter sweeps for a large set of optimization studies and debugging processes.

# 1 Introducing Shock Ignition

In today‘s economic climate, energy is a precious commodity; it is therefore a leading factor in the design of most engineering budgets. In an effort to change this paradigm, many societies have attempted to stray away from using traditional fossil fuels and move towards other forms of large-scale energy production.

One potential source for this energy production is nuclear fusion, the same process that powers the sun. The reason this form of energy production has not been realized in a reactor setting, though, is because a feasible reactor would need to confine an extremely hot, dense plasma. Because these terrestrial plasmas cannot take advantage of the large gravitational forces present in the sun, other schemes for their confinement are needed.

## 1.1 Confining a Plasma using its own Inertia

Inertial confinement fusion (ICF) is a method for achieving fusion that utilizes a plasma‘s own inertia as a form of confinement. In it, a spherical array of high-intensity lasers deposits energy onto a small plastic target filled with Hydrogen isotopes, see Fig. 1.1. The energy deposited by these lasers serves to both heat and compress the target, causing its Hydrogen fuel source to reach the temperatures and the pressures needed for ignition, or the initiation of a sustained fusion chain reaction [1].

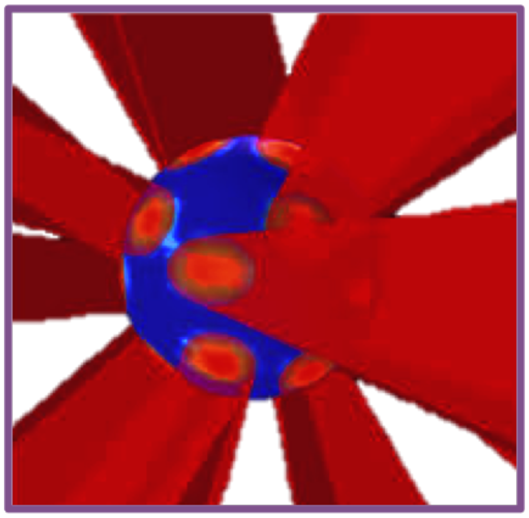


**Figure 1.1:**
**Inertial Confinement Fusion** is sometimes referred to as laser-driven fusion because it involves shooting a millimeter-sized target with an array of lasers. This image captures how the areal projections of the lasers impact the spherical symmetry of the problem [2].

After ignition, the fuel source continues to burn until a dismantling shockwave - originating at the center - has a chance to propagate outwards. Once this shockwave reaches the target‘s shell, it ruptures the surrounding plasma envelope and causes the target to explode. The optimization problem at hand, then, is to maximize the number of fusion reactions that occur in the nanoseconds-length time window between the target‘s ignition and subsequent explosion.

In order to develop a working inertial fusion reactor, though, many implementation decisions and technologies still need to be made. Several research facilities exist to investigate such issues, most notably the one at the National Ignition Facility in Livermore, California (Fig. 1.2). At these facilities, one issue that gets a considerable amount of research attention is the type of ignition.

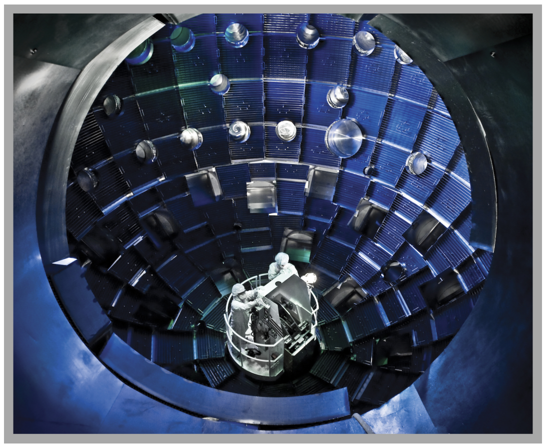


**Figure 1.2:**
**The National Ignition Facility** houses the world's largest laser array. Inside the holes of the spherical chamber are nearly two-hundred lasers. At roughly ten meters, the chamber itself is four orders of magnitude larger than the plastic targets it ignites.

Currently, the two most common types of ignition are direct drive and indirect drive. In direct drive ignition, the laser array focuses directly on a target; while in indirect drive ignition, the laser array focuses on a gold shell surrounding the target that then deposits X-rays onto the target. Although these two types of ignition are the most common, others exist: one prospective type being shock ignition [3].

## 1.2 Achieving Ignition with Aligned Shock Fronts

Shock ignition gets its name from how it achieves ignition: by timing shockwaves so that they add constructively at the center of a fuel target. The reason these shockwaves are able to reach a specific location simultaneously is because of their different speeds.

In shock ignition, shockwaves are produced asynchronously by fluctuations in laser power. Therefore, to have each shockwave coalesce at a certain radius later on, faster and faster wave speeds are needed for each successive shock. Because these wave speeds are positively correlated to laser power, shock convergence can be achieved with a laser profile of individual pulses that increase in power. This then transforms the problem of aligning shockwaves into one of tuning a laser array's power versus time profile [4].

This problem of constructing a laser array's power vs. time profile is the fundamental difference between shock ignition and other leading ignition schemes. In conventional schemes for ignition, laser array profiles are designed to couple the compression stage and the ignition stage of the target, while in shock ignition, the profiles are designed to decouple them [5].

As shown in Fig. 1.3, this decoupling is accomplished by dividing the main laser pulse, which accounts for most of the energy consumption, into two separate pulses: a compression pulse and an igniter pulse. The compression pulse, which is a relatively long, weak pulse, is designed to heat and to compress the target. While, the igniter pulse, which is a relatively quick, high intensity burst, is designed to actually cause the target to ignite. This decoupling of the two pulses leads to shock ignition having several unique characteristics when compared to other forms of ignition.

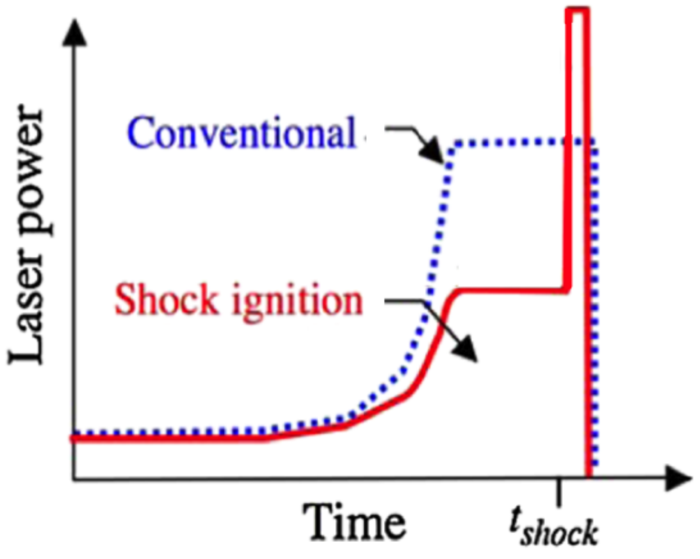


**Figure 1.3: Laser Power vs. Time Profiles for Shock Ignition and Direct Drive Ignition.** Shock ignition differs from Direct Drive ignition in the way its laser profile is constructed. Shock ignition separates what would be the main pulse into two pulses, which consequently decouples the target's compression stage and its ignition stage [5].

The major advantage shock ignition has over other competing ignition schemes is that its targets prospectively produce higher net gains, see Fig. 1.4. Here, a gain is defined as the ratio of fusion energy produced to the driver energy incident on the target, where a higher gain would correlate to a higher return of energy [1].

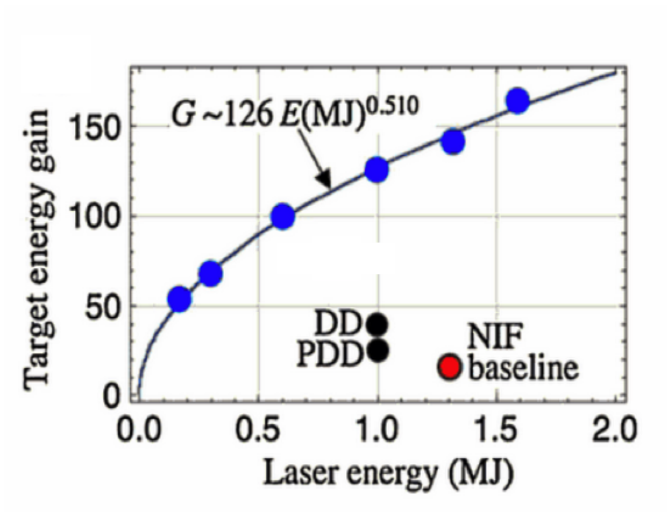


**Figure 1.4:**
**Gain as a function of Laser Energy for Shock Ignition.** Early 1-D simulations show that the gains from shock ignition targets not only exceed the NIF baseline requirements, but also surpass those of candidate targets for both direct drive ignition (DD) and polar direct drive ignition (PDD) by nearly an order of magnitude [5].

Shock ignition is capable of achieving these gains because its targets have longer burn times, a consequence of the relatively low implosion velocities and thick instability-resilient shells inherent to the process [5]. A corollary of these high gains is that a feasible shock ignition reactor could produce the same amount of energy as any other potential inertial fusion reactor with a much weaker, and thus a much cheaper, laser [3].

In order for shock ignition to actually achieve these gains, though, its laser array's power versus time profile has to be constructed properly. This construction process entails combining several laser pulses together in a very specific configuration. Here a laser pulse is a section of the power versus time profile that has some definite shape, such as a horizontal line or a Gaussian curve.

In addition to being constructed properly, a laser profile needs to be constructed efficiently. There are several reasons for this. First, because time at inertial fusion research facilities is expensive, many simulations are needed to justify every set of experiments. Second, because there are so many research branches, e.g. target design and polar drive, the number of simulations needs to be minimized.

# 2 Constructing a Laser Profile

Shock ignition involves aligning shockwaves so that they coalesce at the center of a target. Because these shockwaves are the result of fluctuations in laser power, the laser profile containing these fluctuations must be constructed with a thorough and robust algorithm.

As shown in Fig. 2.1, there are six different laser pulses that constitute a shock ignition laser profile: a picket pulse, three pedestal pulses, a compression pulse, and an igniter pulse. In terms of comparing shock ignition to other forms of ignition, the compression pulse and igniter pulse are the major differences, when it comes to power considerations.

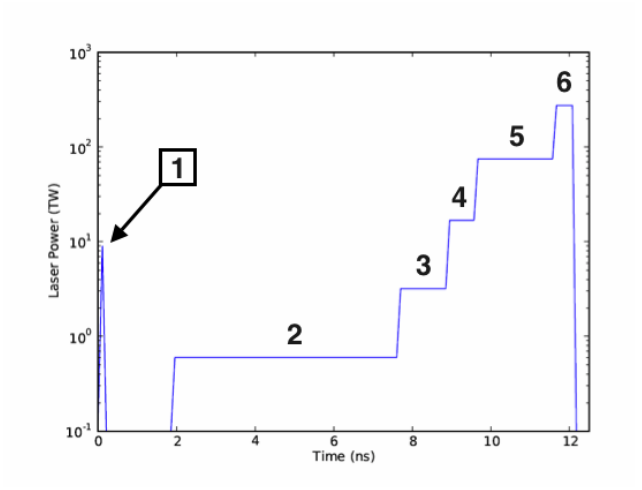


**Figure 2.1: An Example Laser Power versus Time Profile.**
A laser power versus time profile for shock ignition has six individual pulses. It has: an initial picket pulse to develop a plasma envelope surrounding the target, three pedestal pulses to increase the target's density, a compression pulse, and a final igniter pulse [4].

## 2.1 Imposing a Picket Pulse

The picket pulse's main goal is to set the initial conditions of the simulation correctly. Before the target is irradiated, it is a cryogenic piece of Hydrogen enclosed by a plastic shell. The picket pulse is used to develop a plasma envelope surrounding the target.

This plasma envelope is useful for several reasons. First, it increases the uniformity of laser absorption, which increases the spherical symmetry of the problem [6]. Second, it decreases the rate of instability growth on the shell's surface, see Fig. 2.2.

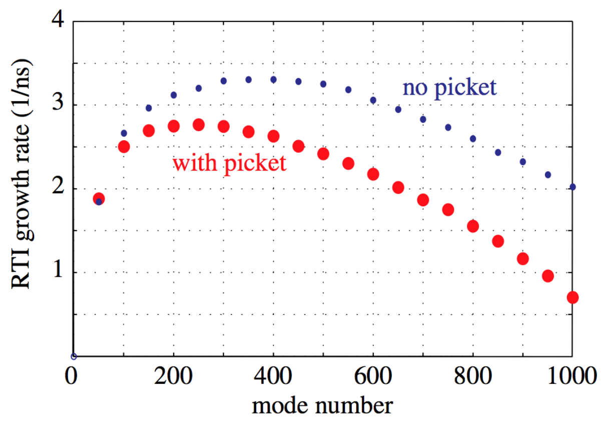


**Figure 2.2: The Picket Pulse** is designed to reduce the growth of Rayleigh-Taylor Instabilities (RTIs). In this figure, the simulated RTI growth rate is taken as the linear growth rate in the 1.5 second window following the convergence of the three pedestal pulses at the ice-gas Hydrogen boundary for a wide range of mode numbers [7].

## 2.2 Setting up Pedestal Pulses

The goal of the pedestal pulses is to increase the target‘s fuel density by a factor of thirty, with a minimal amount of compression in the target. There are three pedestal pulses because each resulting shock is only theoretically capable of increasing the fuel density by a factor of four. Therefore, to achieve the sought after factor of thirty increase in fuel density, a minimum of three consecutive pulses are needed [4].

To maximize the energy gain, the pedestal pulses need to be timed so that their resulting shockwaves arrive at the interface that exists between the outer Hydrogen ice shell and the inner Hydrogen gas region within fifty picoseconds of each other. This interface is the optimum location for two reasons. First, if the shockwaves were to pass each other within the Hydrogen ice region, they would cause unnecessary shock heating of the fuel. Second, if the pulses were to meet inside the Hydrogen gas region, they would cause the fuel to decompress, which would be counter-productive to the process [8].

## 2.3 Optimizing the Compression and Igniter Pulses

After the target‘s density has increased by a factor of thirty, two more pulses are needed: the compression pulse and the igniter pulse. Where constructing the picket pulse and the pedestal pulses required coalescing the resulting shockwaves at a single location, constructing the main pulses involves optimizing certain variables. For the compression pulse, the maximized variable is the target‘s density; for the igniter pulse, the maximized variable is the energy yield [4].

# 3 Modeling Shock Ignition

Before shock ignition experiments can be conducted at research facilities, such as the National Ignition Facility, the algorithm used for tuning the involved laser profiles needs to be both efficient and robust. It needs to be efficient because each experiment involved in an actual tuning process requires more than a hundred simulations [4]. It needs to be robust because the output from the simulations needs to transfer to successfully timed experiments without too much calibration [6].

In this paper, simulations were run on Bucky – a one-dimensional radiation-hydrodynamics code written by Prof. Gregory Moses. Because many simulations were needed, several assumptions were made to reduce their individual runtimes. These assumptions were made because they did not significantly increase the numerical error of relevant quantities.

## 3.1 Using Bucky: A One-Dimensional Radiation-Hydrodynamics Code

Bucky is a one-dimensional, Lagrangian, radiation-hydrodynamics code that models plasmas that can be classified as high temperature and high density. Here, radiation-hydrodynamics refers to the tight coupling that Bucky assumes exists between the radiation and the hydrodynamics for its problems' energy conservation equations [9].

The next term, Lagrangian, describes how mass is always conserved inside a zone and its boundaries are what fluctuate. This is analogous to the distortion that would occur to cross-hatched lines drawn on a slab of stretched rubber.

Bucky's final descriptor as a one-dimensional code refers to the fact that it only accounts for one spatial dimension in addition to its temporal one. Although one-dimensional codes are the most simplistic, three-dimensional codes are the most realistic because they simulate the space in which humans actually exist. This initial comparison does not do Bucky justice, though. Some physical phenomena can be modeled in certain coordinate systems (e.g. Cartesian, spherical, and cylindrical) to take advantage of inherent geometric symmetries.

Because inertial confinement fusion involves a spherically symmetric target, it can be modeled in the spherical coordinate system. Additionally, because the target is ideally compressed isentropically – in a uniform way with the least amount of work – the problem should not initially depend on angle. Because two of the dimensions involved with the spherical coordinate system are angles – the azimuthal angle and the polar angle – the problem can be reduced to one-dimension: the radius [1].

This one-dimensional nature does, however, prevent Bucky from directly studying the higher dimensional effects of instabilities and turbulence; it is therefore best used for simulating the early stages of inertial confinement fusion. This restriction does not completely preclude Bucky from investigating instabilities, though. Several parameters exist than can provide guidance when transitioning output to higher dimensional codes and, even, to actual experimental shots. These parameters include the peak implosion velocity and the convergence ratio of the target's initial radius to its smallest radius [5].

In addition to being markedly simpler than three-dimensional codes, one-dimensional codes are also much quicker. Where a three-dimensional code might take weeks to finish a simulation, a one-dimensional one might take hours. This allows Bucky to run tests that would not be possible on higher dimensional codes.

## 3.2 Asserting Appropriate Approximations

To time shocks using Bucky, it is essential to streamline the process as much as possible by using several well-chosen approximations. Most of these approximations stem from the fact that only a short amount of time is spent at high laser powers and that this time occurs at the end of laser profiles. The most prolific approximation that arises from this is that simulations only require conventional radiation-hydrodynamics [5].

For Bucky, use of conventional radiation-hydrodynamics manifests itself in being able to ignore two radiation effects: transport and preheat. Omitting radiation transport reduces the number of solved equations, decreasing runtimes by more than a third. Disregarding radiation preheat allows the laser profile to be constructed iteratively [4].

The ability to construct laser profiles iteratively allows for a very systematic approach to be taken. The essence of this approach is to mate optimized laser pulses with a set of target designs subject to maximum laser power and energy constraints [5].

This construction process can be further simplified by imposing a piecewise linear scheme for the pulses. If suitable laser powers and durations are then given, the problem reduces to simply timing each laser pulse's beginning.

The last major approximation applied for shock ignition is turning off the thermonuclear burn. Although this particular approximation reduces the accuracy of the simulation, its significant effect on runtime justifies its use. For Bucky, runtimes receive a factor of six reduction, dropping them from nearly two hours to just under twenty minutes. Combined with the other approximations, runtimes of around ten minutes are achievable [4].

# 4 Running Multiple Instances of a Program

In scientific computing, an often-occurring bottleneck is the collection of data. Whether it be the result of running one simulation, two simulations, or many simulations, a particular research problem can easily take days of computer time to run. Therefore, a major goal of many programming languages and software packages is the optimization of computer resources.

The two branches of modern scientific computing concerned with this optimization of computer resources are: high-performance computing (HPC) and high-throughput computing (HTC). The main distinctions between the two is that HPC focuses on decreasing the runtime of a single simulation, while HTC focuses on increasing the throughput of many simulations. In terms of timescales, HPC maximizes the floating operations per second (FLOPS), whereas HTC maximizes the total number of operations done per month or even per year.

In the case of running multiple instances of a program, as is needed for timing shocks, HTC is usually the primary mode of optimization. Practically, this is done when a single instance of a program runs relatively quickly and would, therefore, not receive dramatic increases in throughput from HPC. As this number of program instances approaches the hundreds and the thousands, though, manual modification and simulation becomes intractable. Methods for automation then become needed; this is why AHAB – or Automated HTC Achieved with Batch Processing – was created.

## 4.1 Conducting Parameter Sweeps with AHAB

AHAB – or Automated High-throughput computing Achieved with Batch processing – is a program used to conduct parameter sweeps over other programs' variables. This is done by using a syntax that allows new, independent variables to be created. This is useful in a practical sense for optimization studies and for debugging. For Shock Ignition, AHAB was used to time the three pedestal pulses, see Section 2.2.

A parameter sweep, or a Cartesian product, is a method for creating every combination of a set of variables and their corresponding set of value lists. Fig. 4.1 shows one example of a parameter sweep done over a coin and a six-sided die that produces twelve combinations. Another example is the fifty-two combination parameter sweep a deck of playing cards does over its four suits and thirteen ranks.

In the two parameter sweep examples given so far, only scalar variables were used. For Bucky, a one-dimensional code, the ability to sweep over one-dimensional vectors is also extremely important. This was the motivation behind developing a syntax that allows new, independent variables to be created.

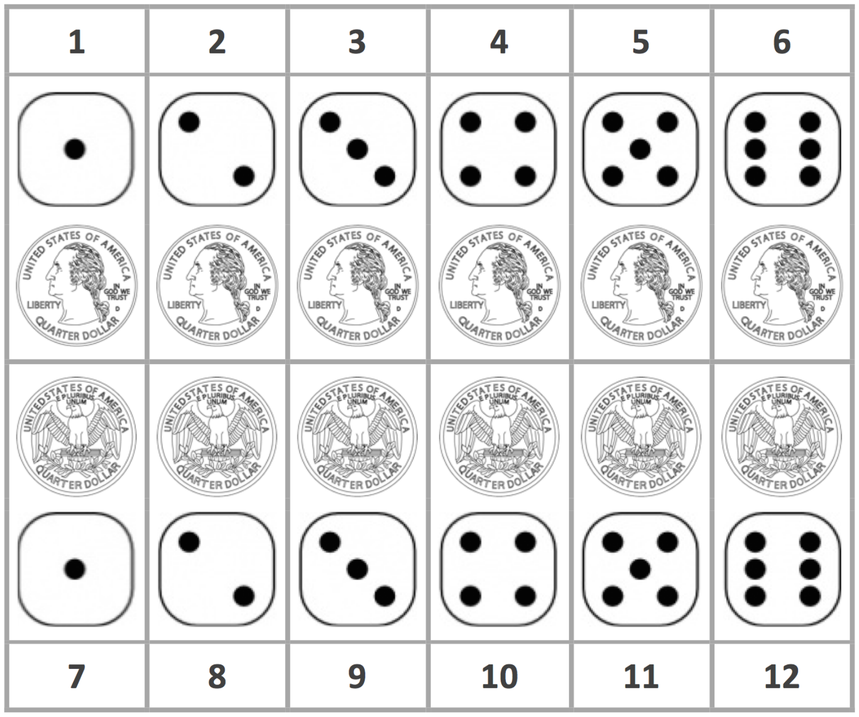


**Figure 4.1:**
**A Parameter Sweep** over a six-sided die and a quarter.

Notice here that both the six-sided die and the quarter exist in a class of objects that have N distinct faces: A quarter has two faces and a six-sided die has six. With AHAB, both these objects could be abstracted to have any multiple number of faces.

To get AHAB sweeping over one-dimensional vectors, though, a syntax and a construct were needed. The syntax was needed to assist with input file parsing (see Fig. 4.3), while the construct was needed to set up the logic of the sweeps (see Fig. 4.2).

AHAB's construct is how it treats variables. This includes the distinction between: scalar and vector, completely new and old, independent values and values in sets. Fig. 4.2 shows a basic example of how this construct would handle three different variables.

The difference between the three variables in Fig. 4.2 is their form of insertion. This notion of insertion characterizes how combinations are created for the parameter sweep. Variable (a) and variable (b) use the simpler insertions, whereas variable (c) uses a combination of the two.

Notice in Fig. 4.2 that variable (a) does not create a new combination, this is why dual insertion works: at each step of its vertical insertion process, a horizontal insertion takes place. Although the figure suggests that vertical insertion can only exist in one dimension, it can in fact take up any number of them.

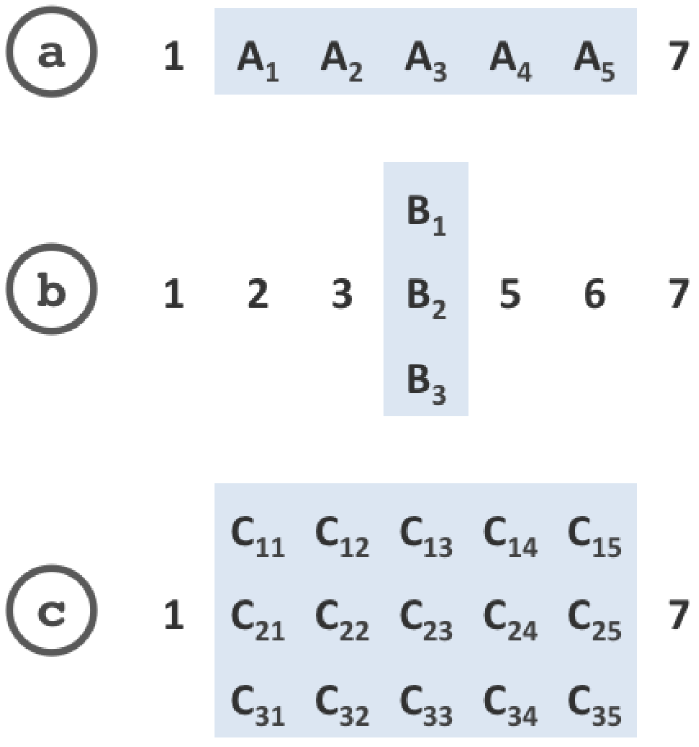

a
1 $A_1$ $A_2$ $A_3$ $A_4$ $A_5$ 7
b
1 2 3 $B_1$ $B_2$ $B_3$ 5 6 7
c
1 $C_{11}$ $C_{12}$ $C_{13}$ $C_{14}$ $C_{15}$ $C_{21}$ $C_{22}$ $C_{23}$ $C_{24}$ $C_{25}$ $C_{31}$ $C_{32}$ $C_{33}$ $C_{34}$ $C_{35}$ 7

**Figure 4.2:**
**A Parameter Sweep** over three 1-D vectors.

**(a)** uses horizontal insertion to construct large vectors.

**(b)** uses vertical insertion to create a new vector for each value of B.

**(c)** uses dual insertion to construct many large vectors. It does this by using new, independent variables to combine both horizontal insertion and vertical insertion.

To accomplish dual insertion, a method to add new, independent variables was needed. In AHAB, these variables are called constants. Fig. 4.3 shows how two constants, **t0** and **tDeltaFrac**, can be used to set up several vectors: *tn2c*, *te2c*, and *tr2c*.

Fig. 4.3 also demonstrates several forms of syntax used by AHAB. The **"@"** is used for constants. The **"!"** and **"#"** are used for comments. The **";"** and **"|"** are used for incrementation. And the brackets are used to distinguish vectors. This only scratches the surface of AHAB's syntax, but gives a good overview of its use of special characters.

Lastly, notice that there are several ways to set values. in Fig. 4.3, semicolons are used to do logarithmic spacing between two values; while pipes are used to linearly increment values starting at a certain number. This allows large lists to be constructed concisely.

```
# -------------------------------------
#   init ; final ; numVals // ( log )
# -------------------------------------

  @t0         !  initial temperature

       1e-2 ; 1e+2 ; 3

  @tDeltaFrac !  temperature fraction

       100  ; 1000 ; 2

# ---------------------------
#  init | increment | numVals
# ---------------------------

  tn2c = [ ( @t0 | @"t0/tdeltafrac" | 250 ) ]
  te2c   [ ( @t0 | @"t0/tdeltafrac" | 250 ) ]
  tr2c = [ ( @t0 | @"t0/tdeltafrac" | 250 ) ]
```

**Figure 4.3: An Example AHAB Input File** originally used for debugging.

Its goal was to test a new physics model for Bucky on a wide range of temperatures. This was done by setting the initial temperatures for the 250 zones' ions, electrons, and radiation.

Here, each temperature vector would start at a value of t0 and be increased two hundred and fifty times in equal increments of t0 divided by the current value of tDeltaFrac.

## 4.2 Utilizing HTCondor and a Network of Computers

As the number of combinations from parameter sweeps increases, it becomes unwieldy to run every simulation on one computer. To remedy this problem, AHAB uses HTCondor as a method to distribute its simulations across a network of computers. This required Bucky's start mode capabilities to be redesigned, see Appendix B.

**Figure 4.4: HTCondor** allows users to utilize a computer network's underutilized cycles.

HTCondor is an open-source project used for high-throughput computing (HTC). It was created in 1984 by Miron Livny at the University of Wisconsin - Madison and was heavily developed by Michael Litzkow. Currently, HTCondor is used at NASA, CERN, and several dozen universities [10].

HTCondor works by matching a user with a specific computer on their network. Due to HTCondor's persistent motto to "leave the owner in control, regardless of the cost," matched computers are ones qualified as low-activity [10]. On a network integrated into a computer lab, this practically means that simulations run fastest at night.

When simulations are run during peak-use times, they will frequently be lifted off busy computers and restarted on unused ones. To allow this process to behave robustly, HTCondor offers the "Standard Universe." This mode does, however, require a program to be compiled directly with HTCondor. When it is difficult or impossible to do so, the "Vanilla Universe" must be used.

Bucky uses the "Vanilla Universe" because its CMake file does not easily allow for the HTCondor compiler to be used. In terms of development, this means that Bucky must be able to handle automatic restarts without the loss of data. This was why the Automatic Mode was created in Bucky, see Appendix B.

# 5 Conclusion

The initial goal of this research project was to continue the work of Dr. Matt Terry's thesis by reconstructing his automatic tuner and then investigating target design. Over time, however, this project has transformed into a more automation focused one, spurred on by the many common computational tasks involved in shock timing.

This departure from creating a single-purpose automatic tuner used for investigating target design has led to many tools with relevance outside of shock ignition and even of inertial fusion. AHAB, for example, was constructed because the original automatic tuner was unsalvageable. Although its use for this project was limited to timing the picket pulse and the pedestal pulses (see Fig. 5.1), it could be used to automate the entire shock timing process. This would increase the scope of research problems to ones requiring multiple tuned laser profiles, such as the design of fuel targets.

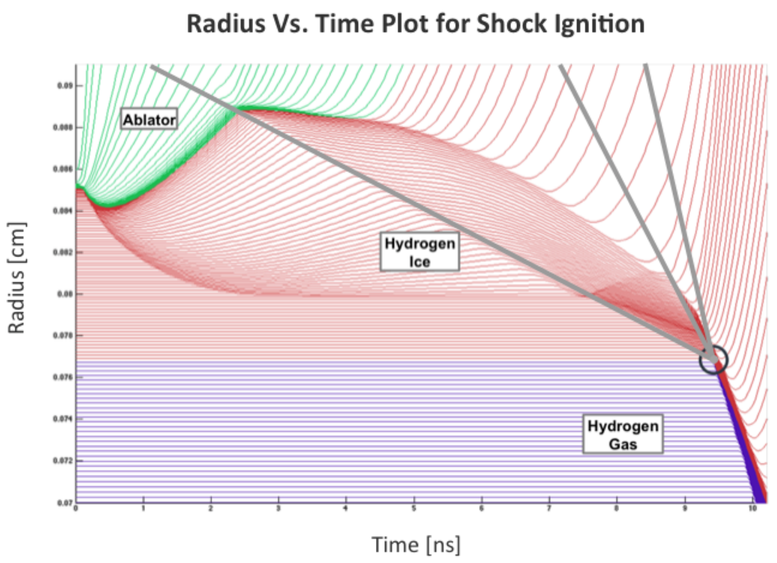


**Figure 5.1: The Convergence of the Three Pedestal Pulses** at the ice-gas boundary of the Hydrogen fuel source. This type of plot is discussed in Appendix A. The main point to take from it is that on a Radius Vs. Time plot, the three drawn lines represent the resulting shocks' wave speeds.

In order to develop an automatic tuner, though, another program would be needed. This program, currently codenamed Mobile Operating Batch Inspector (MOBI), would be sent to HTCondor and would then submit multiple instances of AHAB. Further, because laser profile construction is iterative, each new laser segment could be written to its own input section, see 'Loading Multiple Input Decks from One File' in Appendix B.

The first step in developing this automatic tuner would be to quantify the accuracy of shock timing. This could be done by introducing a new parameter – the shock breakout – that could be inspected by MOBI in post-processing to guide future AHAB submissions. This would prevent the need for human interpretation and interaction, both of which were used to construct the profile in Fig. 5.1.

For timing the shocks, the term *shock breakout* refers to when the outermost Hydrogen gas layer reaches a velocity of 3.5e5 cm/s [4]. Using this method, each new shock would have its shock breakout time plotted against its launch time. Because a late-arriving shock would not alter this breakout time, the optimal time to start would, then, be when the breakout time just starts to move, see Fig. 5.2.

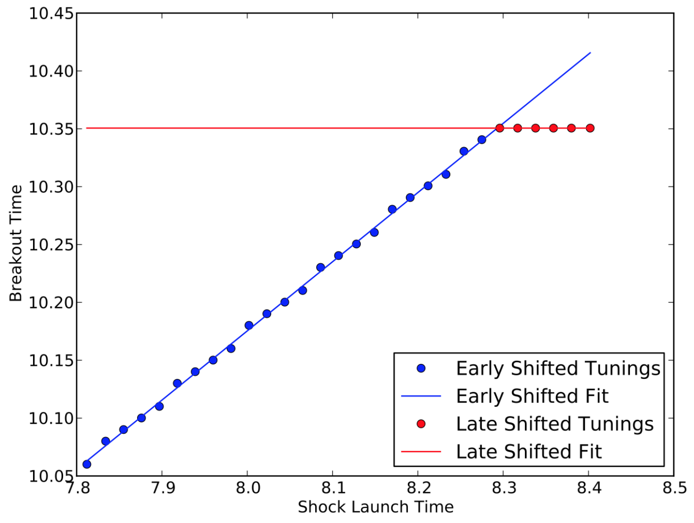


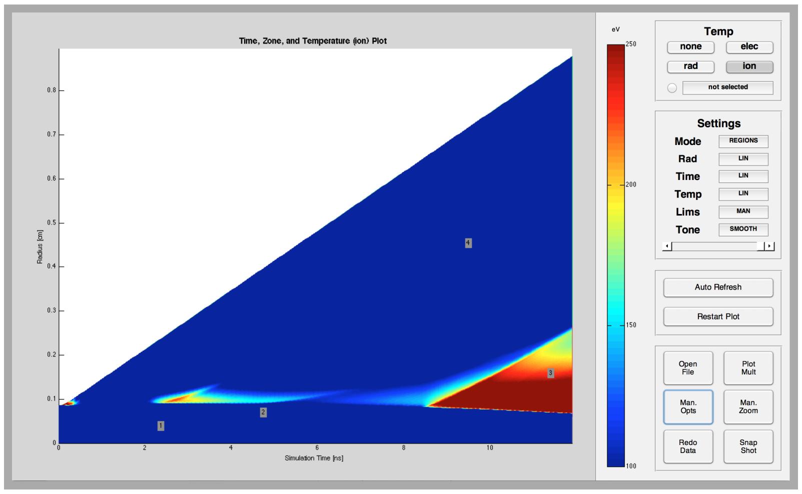
**Figure 5.2: Timing a Pedestal Pulse** requires finding the intersection between the positively sloped region of the breakout time curve and the horizontal region of it. This is done by varying the shock launch time and measuring the resulting breakout time [4].

As mentioned in Sec. 2.3, the compression pulse and the igniter pulse require optimizing some variable. Although these optimized variables were stated as the target density and the energy yield, respectively, each optimized quantity could be further complicated by investigating multiple properties simultaneously. This could include: increasing a pulse effect's robustness, achieving certain value requirements, and developing new parameters to quantize certain physical behaviors [4; 5; 7].

Although some other unforeseen tools and analytics will be needed, AHAB should now be in a state that allows integration into a program like MOBI without too much modification. To make sense of the Bucky output along the way, the Radius, Time, and Temperature Plot discussed in Appendix A will prove to be invaluable.

# A : The RTT Plotting Function

The Radius, Time, and Temperature (RTT) Plotting Function is a tool used to create common plots for 1-D Lagrangian Radiation-Hydrodynamics codes, like Bucky. The Radius and Time refer to the ordinate and the abscissa of the 2-D plot, respectively, while the Temperature is in reference to a 2-D Variable's color plot that is overlaid on top of the underlying Radius Vs. Time plot (see Fig. A.1). The Radius Vs. Time Plot is the fundamental plot for 1-D Lagrangian Codes because it represents its two computational dimensions: space, by radius, and time.



| | |
|---|---|
| Mode | REGIONS |
| Rad | LIN |
| Time | LIN |
| Temp | LIN |
| Lims | MAN |
| Tone | SMOOTH |

**Figure A.1:**
**The RTT Plotting Function** has an intricate graphical user interface that is designed to give researchers a great degree of freedom when exploring data. The sidebar shown on the right has four sections of buttons. The first two sections are involved in changing the plot on the left, while the bottom two deal with special utilities.

## A.0 Temperature Selection

The topmost section of buttons on the GUI is involved in changing the overlaying temperature plot. This connotation of overlaying refers to how temperature variables in Lagrangian codes have values at every spatial position and every temporal position.

For Bucky, the three temperatures that are most relevant are: the electron temperature, the ion temperature, and the radiation temperature. However, this notion of temperature was later redefined by extending it to include all variables that have a value for every zone and time. This can be done by pressing the button labeled 'not selected'.

## A.1 Temperature Tone

The bottommost button in the 'Settings' section is the Tone button. It changes how the Temperature overlay plot is colored. The default setting of 'Tone' is 'Smooth'. Smooth performs an averaging scheme to its discrete data to make it look continuous. This can prove to be dangerous in some situations because the averaging may be ill-justified. In those situations - most commonly with logarithmic scales - it is useful to switch the 'Tone' to 'Flat'. This mode involves no data manipulation.

## A.2 Plot Mode

The Plot Modes button is the first button in the 'Settings' section. It refers to how the base Radius Vs. Time plot is done, see Fig. A.2. When Mode = 'Regions', the different layers of material are colored as blocks and labeled with numbers. When Mode = 'Lines', each Lagrangian cell boundary is plotted with region labels attached.

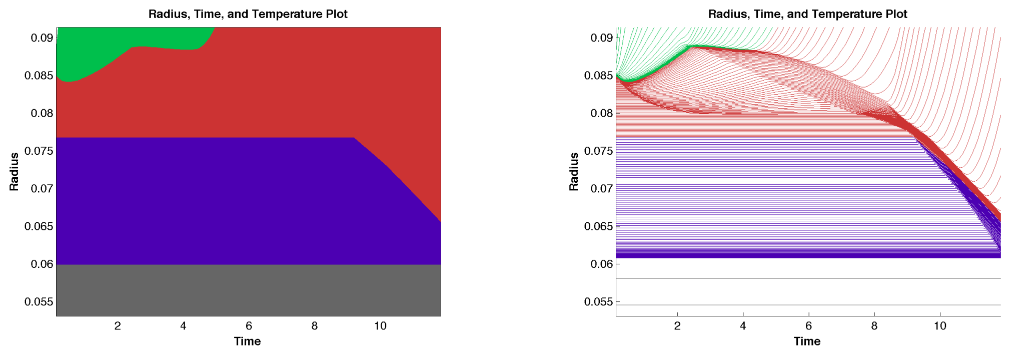


**Figure A.2: Comparison of the Regions and Lines Plotting Modes.**

## A.3 Dual-Function Slider

The Slider on the bottom of the 'Settings' section is related to the plot mode button. When the Mode = 'Regions', the slider changes the transparency of the temperature overlay plot, where the *rightmost* value is equivalent to if the base Radius Vs. Time Plot is completely hidden and the *leftmost* value is equivalent to if 'none' is selected in the 'Temp.' section. When the Mode = 'Lines', the slider changes the number of zone boundary lines that get shown.

## A.4 Scales

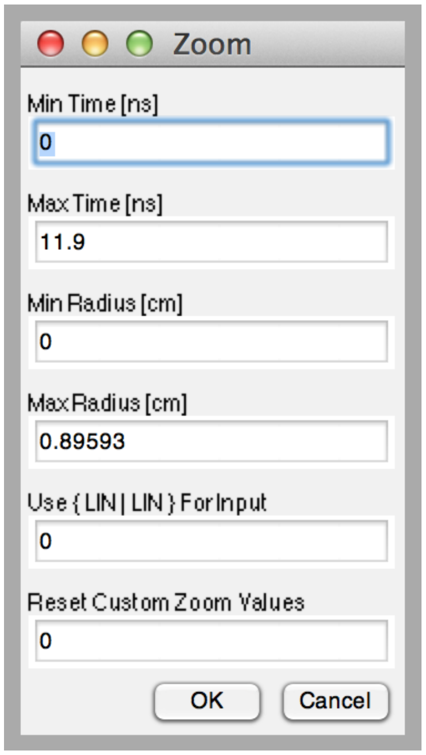
The RTT function has three buttons in the 'Settings' section that change the scaling of the problem. Initially, all three scales - radius, time, and temperature - are set to linear. They can each be changed to logarithmic and back by clicking their respective buttons.

## A.5 Manual Limits

In addition to changing the scaling of the problem, another common plotting problem is imposing certain limits (i.e. maxima and minima). This can be done for the radius, the time, and the temperature; however, the temperature is handled differently.

The Temperature limits involves two buttons that are combined for the other variables. The first button is 'Man. Opts', which changes the maximum and minimum temperature, while the second button is 'Lims' in the 'Settings' section. Initially, the 'Lims' button is set to 'Auto', but can be changed to 'Manual'. The 'Manual' setting changes the temperature limits from the automatically chosen ones to the values inputted in 'Man. Opts'. By default, these limits are 0.1 eV and 200 eV.

For Radius and Time, the manual limits are controlled with the 'Man. Zoom' button. The term manual refers to how it differs from the automatic zoom that is imposed on the plot at the beginning of the program. Manual zooming is, thus, changing the minimum and maximum values for the radius and for the time. With the input box, this can be done in both the current scale and the Linear-Linear scale. Additionally, the input box allows a way to restart the extrema to their initial values, which are in turn set by the current file's data, see Fig. A.3.



Min Time [ns]
0

Max Time [ns]
11.9

Min Radius [cm]
0

Max Radius [cm]
0.89593

Use { LIN | LIN } For Input
0

Reset Custom Zoom Values
0

**Figure A.3: The Zoom Input Box** gives users a way to zoom to specific areas of the plot. Changing "Lin|Lin" forces these values to be linear, while changing "Reset 'Custom Zoom Values" resets them.

## A.6 Loading and Reloading Data

Several buttons in the bottom two sections provide loading and reloading functionality. ′Open File′ opens a prompt for users to select a new file to load. This function is called at the beginning of the RTT function, either implicitly if a data file exists in the local directory, or explicitly with a prompt if no data file is found.

The ′Redo Data′ and the ′Auto Refresh′ buttons offer two ways to reload data. The ′Redo Data′ button reopens the current data file. This is useful for when a run is restarted. The ′Auto Refresh′ reopens the current data file ′n′ times at regular intervals of ′m′ seconds. This is useful for when the RTT plot is being used while a Bucky simulation is simultaneously running. The values of ′n′ and ′m′ are set inside ′Man. Opts′ with default values of 10 seconds and 500 times, respectively.

The ′Restart Plot′ button is used to return every selection in the top three sections to their original values. It is located inside the same section as ′Auto Refresh′ because it is the only button able to turn off the ′Auto Refresh′ effect – besides by pressing of the ′Auto Refresh′ button again.

## A.7 Saving the Current Plot to a File

The ′Snap Shot′ button allows researchers to save the current plot to an image. This is commonly needed for published documents and presentation slides. This routine has a user select a file name, a destination, and an image type and then saves the plot there. This function most notably increases the font sizes and removes the background color.

## A.8 Plotting Multiple Files

The ′Plot Mult′ button allows users to open a whole directory of data files and displays their contents inside a scrollable window, see Fig. A.4. The plots produced by this will all share the options set by the user in the top two button sections of the main plot.

Additionally, when the directory is created by AHAB, every plot′s zoom feature is connected together. This allows users to see the subtle differences in their data (e.g. the arrival time of a specific shock at a certain zone).

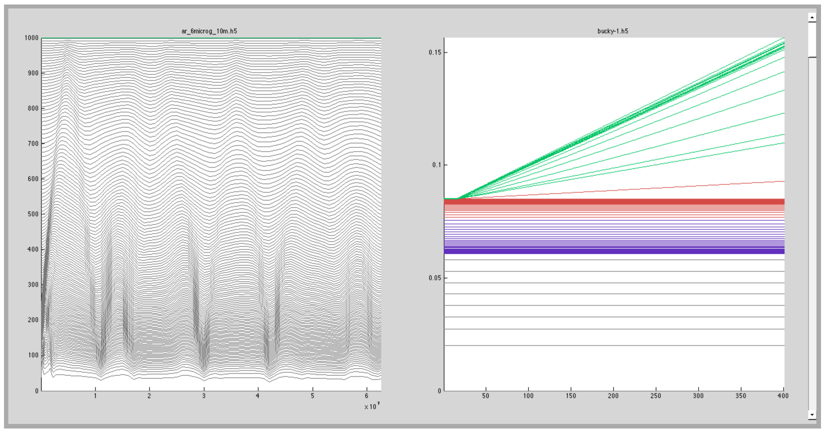
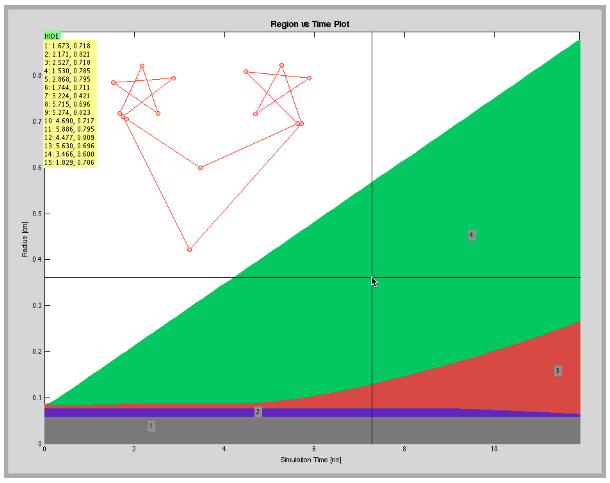


**Figure A.4: The 'Plot Mult' Button**
plots multiple files inside one window.

## A.9 Adding Points to the Plot

The last feature the RTT plot offers is the "Points Layer." In the RTT plotting function, each plot is given its own layer (i.e. the Radius Vs. Time Layer, the Temperature Layer, the Text Layer). The Points Layer is another layer that allows users to add points to the current plot. These points transform under scale transformations and zooming.

Additionally, while hovering over the plot in the GUI, a crosshairs is generated; this crosshairs is composed of a vertical line and horizontal line that meet at the cursor and extend to the boundaries of the plot. When any key besides 'Delete' or 'Escape' is pressed without a modifier (i.e. 'Shift'), a point appears on the plot and has its linear scale coordinates entered on the top left of the screen. Any number of points can be added to the plot; however, they are all removed when opening a different file.



```
1: 1.673, 0.718
2: 2.171, 0.821
3: 2.527, 0.718
4: 1.530, 0.785
5: 2.868, 0.795
6: 1.744, 0.711
7: 3.224, 0.421
8: 5.715, 0.696
9: 5.274, 0.823
10: 4.690, 0.717
11: 5.886, 0.795
12: 4.477, 0.809
13: 5.630, 0.696
14: 3.466, 0.600
15: 1.829, 0.706
```

**Figure A.5:**
**The Points Layer**
allows points to be added to the current plot. This is done by pressing most keyboard keys while hovering over the plot. These points persist through scale changes, but are removed when the file is changed. The coordinates are displayed in the top-left of the plot.

# B : Bucky Start Mode Capabilities

The Bucky start mode capabilities allow users to stop and restart simulations at whim. This allows minor changes to be made to the code or the input file without the need for repeating the whole simulation. This functionality is controlled by the user using the two namelist restart variables: **mstart** and **ncycrs**, which specify the start mode and the cycle number, respectively (see Fig. B.1).

```
$rstart
         mstart  =  2
         ncycrs  =  200
$end

$input
         nmax    =  1300
         tmax    =  5.0E-09
         iorest  =  500
$end
```

**Figure B.1:**
**Example 'bucky.inp' File**
that shows the common structure for using the start mode functionality.

Inside the **$rstart...end$** block, mstart and ncycrs are changed.
Inside the **$input...end$** block, below that, every other variable is changed.*

*For this specific example, the input deck is short because it is for a restart run that does not need to specify the initial conditions of the problem.

## [ 0] – Standard Mode

In Standard Mode, when mstart = 0, the current simulation is treated as an initial run. This means that it will overwrite output files, e.g. bucky.h5 and bucky.out. It also means that the code treats its input file as if it had no restart block at all.

## [ 1] – Restart Mode

In Restart Mode, when mstart = 1, the current simulation starts from a cycle found in the restart logs of either **bucky.cur_restart** or **bucky.new_restart**. Restart logs are created at every ***iorest*** cycles when the code is in either the standard mode or the restart mode.

The cycle that the restart mode loads from is the closest one to **ncycrs** in either bucky.cur_restart or bucky.new_restart. This means that if ncycrs is greater than or equal to the first log in bucky.cur_restart, the algorithm chooses the closest cycle in either of the two files and loads from it. Additionally, if ncycrs is less than or equal to zero, the code short-circuits to loading the newest log in bucky.new_restart.

When starting a new restart, two things are done. First, if the current log was found in bucky.new_restart, bucky.new_restart is relinked as bucky.cur_restart before being read. Second, the base names of the output files are changed to reflect an advancement in restart number. For example, after the initial run, the first restart changes the text output file from bucky.out to bucky1.out; on the second restart, the text output file changes from bucky1.out to bucky2.out.

## `[ 2]` – Automatic Mode

In Automatic Mode, when mstart = 2, a simulation is allowed to efficiently advance to completion, even in the face of frequent and unprompted restarts. This mode was originally designed for HTCondor's vanilla version, which is discussed in Section 4.2. The main difference between Restart Mode and Automatic Mode is that Restart Mode has files filled with restart logs written every iorest cycles, while Automatic Mode uses checkpoints updated every iorest cycles.

This difference in nomenclature stems from the fact that checkpoint files only have one restart log a piece. Bucky therefore keeps three checkpoint files in Automatic Mode: a backup file with the initial conditions of the simulation set (bucky.restart_0) and two leapfrog files that have their logs updated every other time (bucky.restart_1 and bucky.restart_2). Additionally, if bucky.restart_0 has either been corrupted or deleted, a simulation in this mode will start over from the beginning.

## `[10]` – Merge Mode

In Merge Mode, when mstart = 10, Bucky merges all the output files made by Restart Mode into the main output files (up to the **ncycrs** cycle or to the most recent cycle if specified, otherwise). This means that bucky1.h5 , … , buckyN.h5 as well as bucky1.out , … , buckyN.out are merged into bucky.h5 and bucky.out , respectively.

After this mode, the restart logs are wiped and an updated restart file is made in their place. This is why Automatic Mode calls it after it has completed its simulation. Automatic Mode does not back up its files like Restart Mode does because it has no way to go back more than two checkpoint logs ago, rendering labeled output files useless.

## Loading Multiple Input Decks from One File

Bucky also has the capability of loading multiple input decks from within one file. This is useful because it removes the need to either use multiple input files or edit a single file, both of which are extremely prone to human error. The multiple input deck capability also gives many different spots to restart a simulation, as long as it was not merged along the way.

When there are multiple input decks in one file, they are read sequentially. However, if the first deck is in Standard Mode or in Automatic Mode, it must be kept at the bottom. This is because these decks contain the initial conditions of the problem and are consequently much longer. After this first read, though, the input pointer returns to the top of the file and works its way down.

For runs not taking advantage of the Automatic Mode, using multiple input decks in one file behaves as if there were N input files read one at a time. This means that mstart can only be equal to: 0, 1, and 10. This protocol refers to having one Standard Mode Run, followed by any number of Restarts and Merges given in any order.

When Automatic Mode is used with multiple input decks, restarts can take advanced savings in storage space, speed, and human error. This is increasingly true for longer simulations and larger input files where early cycles become decreasingly important.

To use Automatic Mode with multiple input decks, set all input decks that are no longer the focus of interest to Automatic Mode. This must include the initial Standard Mode run and continue until the first Restart Mode. It is important to note that after transitioning from Automatic Mode to Restarts and Merges, there is no going back. For this mode, mstart can only be equal to: 2, 1, and 10.

# References

[1] J. J. Duderstadt and G. A. Moses, *Inertial confinement fusion*. Wiley New York, 1982.

[2] D. Batani *et al.*, "Physics issues for shock ignition," *Nuclear Fusion*, vol. 54, no. 5, 2014. [Online]. Available: http://stacks.iop.org/0029-5515/54/i=5/a=054009

[3] L. J. Perkins *et al.*, "On the Fielding of a High Gain, Shock-Ignited Target on the National Ignition Facility in the Near Term," Lawrence Livermore National Laboratory, Livermore, CA, Tech. Rep., 2010. [Online]. Available: http://dx.doi.org/10.2172/1013209

[4] M. R. Terry, "Effect of different charged particle stopping power models on ICG ignition," Ph.D. dissertation, The University of Wisconsin - Madison, 2010.

[5] L. J. Perkins *et al.*, "Shock Ignition: A New Approach to High Gain Inertial Confinement Fusion on the National Ignition Facility," *Phys. Rev. Lett.*, vol. 103, Jul 2009. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevLett.103.045004

[6] M. R. Terry, L. J. Perkins, and S. M. Sepke, "Design of a deuterium and tritium-ablator shock ignition target for the National Ignition Facility," *Physics of Plasmas*, vol. 19, no. 11, 2012. [Online]. Available: http://scitation.aip.org/content/aip/journal/pop/19/11/10.1063/1.4765354

[7] S. Atzeni, A. Schiavi, and A. Marocchino, "Studies on the robustness of shock-ignited laser fusion targets," *Plasma Physics and Controlled Fusion*, vol. 53, no. 3, p. 035010, 2011. [Online]. Available: http://stacks.iop.org/0741-3335/53/i=3/a=035010

[8] H. F. Robey *et al.*, "Shock timing experiments on the National Ignition Facility: Initial results and comparison with simulation," *Physics of Plasmas*, vol. 19, no. 4, 2012. [Online]. Available: http://scitation.aip.org/content/aip/journal/pop/19/4/10.1063/1.3694122

[9] J. J. MacFarlane, G. A. Moses, and R. R. Peterson, "BUCKY-1–A 1-D Radiation Hydrodynamics Code for Simulating Inertial Confinement Fusion High Energy Density Plasmas," *UWFDM-984*, 1995.

[10] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the Condor experience," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 2-4, 2005. [Online]. Available: http://dx.doi.org/10.1002/cpe.938