1 Introduction

N-body systems can be characterized by two basic scales that are significant in estimating their run times as well as categorizing the methods that are best applied to them. These scales are the number of bodies needed to explore the relevant dynamics and the length of the simulation time in units of dynamical times. A dynamical time is the significant timescale for some systems in question. For example, in the case of astronomical systems, the dynamical time is the orbit period of the bodies; and the shortest orbital period typically constrains the length of the time step that can be used by the integrator. There are many problems, such as those in the area of cosmology, where the number of dynamical times, \( T \), that one must simulate is quite small \( (T \approx 100) \). As a result, these systems can be explored with very large particle counts \( (N \approx 10^3 - 10^{10}) \). The choice of integrators for these systems is also fairly flexible because one doesn't have to worry as much about compounding errors for such a small number of time steps. On the other end of the spectrum are problems such as those related to the formation of planets or the evolution of planetary and small body orbits. These problems typically require much longer integrations \( (T > 10^7) \) and are currently limited to fairly small particle counts \( (N \approx 100-1000) \). The long time scales required for these integrations also puts significant constraints on the integrators that are used. There are basically two routes to take when integrating these systems. One can either use an integrator that solves the system to machine accuracy \([1]\) or use a symplectic map \([2,3,4]\).

Integrators that operate at machine accuracy might be ideal in regards to accuracy, but they aren't generally the fastest option and often require very small time steps. Given that these are integrations that must run for many millions of dynamical times, using a slow integrator isn't an option. The alternative is to use a symplectic map for the integration. These are methods that work specifically with Hamiltonian systems and have a number of helpful properties \([5]\). The most significant is that they conserve a surrogate Hamiltonian to machine precision. As long as the surrogate Hamiltonian is bound to the real Hamiltonian this leads to conservation of the real Hamiltonian over long time integrations. A symplectic map will provide this benefit without having to be taken to machine accuracy.

Astronomical systems aren't the only instances of Hamiltonian systems that need to be integrated for many dynamical times and that could benefit from having larger numbers of particles. Molecular dynamic simulations are also Hamiltonian and it takes many dynamical times to cover any significant period of real time. Some significant problems from molecular biology, such as doing atomic level simulations of metabolic pathways or even entire cells, would benefit from more efficient symplectic maps \([6]\).

Unfortunately, all existing symplectic maps are \( O(N^2) \) or worse for N-body integrations \([7]\). This imposes a restriction on the number of particles that can be used when studying systems with these integrators. The best existing symplectic maps gain speed advantages by using longer time steps \([8]\), but these can fail for dense systems and the ways of getting around this problem often lead to worse scaling with the number of bodies \([9]\).

This paper presents a significant step toward the development of a symplectic method that scales better with the total number of particles. The method is based on the concepts of tree-codes \([10]\). Standard tree methods break symplecticity for a number of different reasons, as is discussed below. This method uses the tree to define a Hamiltonian that can be integrated using a standard symplectic map. The map then conserves the tree-based Hamiltonian. Using this method, the changes to particle positions and velocities are calculated off the tree and the \( O(N^2) \) cost of standard particle-particle force evaluations are avoided. When fully developed this advance could allow for the numerical exploration of a new class of problems that require many dynamical times and large numbers of particles.

The next section presents a brief review of Hamiltonian systems and symplectic maps. After that is a summary of current tree-based methods and why they are not symplectic.
Section 4 presents our method of defining a tree-based Hamiltonian and one approach to integrating it. This is followed by results using this method and the directions of our future work.

2 Hamiltonian Systems and Symplectic Maps

The Hamiltonian formalism was developed as another way of describing Newtonian mechanics. It produces two first-order differential equations for each dimension of the systems phase space [11]. It is often applied outside of classical mechanics and is useful in quantum mechanics as well [12].

To use the Hamiltonian formalism all we need is the Hamiltonian, \( H(q,p) \), of the system as a function of the generalized coordinates and momenta. In most cases this is simply the total energy of the system. Transforms can be applied to pick coordinates that make the system easier to work with. This isn't required for this work and standard Cartesian coordinates are sufficient.

From the Hamiltonian one gets the time derivatives of the coordinates and positions with Hamilton's equations:

\[
\dot{q}_i = \frac{\partial H}{\partial p_i} \\
\dot{p}_i = -\frac{\partial H}{\partial q_i}
\]  

(1)

A brief example of how this works can be given in the idealized case of a mass, \( M \), on a spring with Hook's constant, \( k \). In this 1-D system the Hamiltonian is given as follows:

\[
H(q,p) = \frac{p^2}{2M} + \frac{kq^2}{2}.
\]

(2)

Applying Hamilton's equations gives the time derivatives for the position and momenta that one would expect for such a system.

\[
\dot{q} = \frac{p}{M} \quad \dot{p} = -k q.
\]

(3)

A symplectic map is a discretization of a system that is area preserving and, for Hamiltonian systems, conserves all the Poincaré invariants [5]. A mapping for the \( q \) and \( p \) vectors can be defined as:

\[
\begin{pmatrix}
 q_{i+1} \\
 p_{i+1}
\end{pmatrix} = L \begin{pmatrix}
 q_i \\
 p_i
\end{pmatrix}
\]

(4)

Such a mapping is symplectic if it satisfies the condition that \( L^T J L = J \) for

\[
J = \begin{pmatrix}
 0 & I \\
 -I & 0
\end{pmatrix}
\]

(5)

Exact solutions for integrable Hamiltonians are always symplectic and the composition of symplectic maps will be symplectic. The way that symplectic maps work is that they provide exact solutions to a system with a Hamiltonian that is perturbed from the actual Hamiltonian being integrated. Generally this perturbed, or surrogate, Hamiltonian remains roughly bound to the original Hamiltonian and oscillates about it. In this way systems can be integrated for very long times with only minor drifts in the total energy of the system, even when lower order mappings are used.

3 Tree Methods

Not all N-body systems require symplectic integrators for their study. Many problems can be solved with only a few dynamical times of integration. For these problems, integrators that don't conserve energy or volumes in phase space are acceptable and a variety of methods can be applied that improve the scaling with particle count. The most commonly used method of improving the scaling with particle count is through the use of tree codes. The concept of the tree code was developed by Barnes and Hut [10]. The idea behind the tree codes is that while interactions with nearby particles need to be done at a particle-particle level, more distant interactions can be approximated by having the particle interact with the potential from a large collection of bodies.

The Barnes and Hut tree method used an octree where the simulation volume is a top level node and each node can recursively be split into eight children. This is done until each node contains only a single particle. Since the original method was published, many variations of the tree-code have been developed that use a variety of different spatial trees [13]. The fundamental concepts of all of these are similar however, and they typically all scale as \( O(N \log N) \).

Another tree based method that has come to be used fairly broadly is based on the fast multipole method, FMM, introduced by Greengard and Rokhlin [14]. These methods evaluate the force on particles by approximating the potential field. The method is significant because when implemented with an adaptive tree the force evaluations can be done in \( O(N) \) time [15]. Building the tree still requires \( O(N \log N) \) time however. The accuracy of FMM methods can be improved by using higher order expansions of the field.

Unfortunately, standard tree methods and multipole methods are not symplectic for a number of different reasons, despite occasional claims to the contrary [16]. Because of the way that basic tree methods have particles interact with collections at a distance, these tree methods don't always conserve momentum because there is no guarantee that the forces will be symmetric. Some techniques have been found that bypass this problem to ensure conservation of momentum [17]. In these cases there is still another problem. The forces that are being applied are not part of a single Hamiltonian and as the particles move about, the structure of the tree changes. This leads to discontinuous jumps in the system which would break the symplectic nature of the integrator, even if the other problems could be overcome.

Since the current symplectic integrators are all \( O(N^2) \) and no existing tree codes are symplectic, certain classes of problems that would require both large numbers of particles and many dynamical times of integration are simply out of...
reach. Examples of this include the early stages of planetary formation and longer metabolic pathways in cells. We now turn to our solution to this problem.

4 Method

The method proposed here is to use a tree, very similar in nature to the original method of Barnes and Hut, but instead of using that tree to explicitly calculate forces, we use it to define a Hamiltonian that can be evolved using standard symplectic integrators. For our purposes, a simple leap-frog or T+V integrator that relies on the separability of the Hamiltonian is used, largely because of its simplicity [18].

The method begins with a complete tree. To aid with the discussion and to keep the math simpler, we use a quadtree instead of an octree. The use of 2-D spatial trees is fairly standard in planetary science problems where many systems have a much larger scale in two dimensions than in the third. Everything that is presented can be easily extended to a third dimension. The number of levels in this tree can be determined by the user.

This method introduces a few new techniques. The goals of these techniques are twofold. First, they need to ensure the algorithm has a Hamiltonian defined from the tree. Second, the Hamiltonian must vary smoothly during the course of normal evolution of the system, otherwise the integrator will be unable to preserve that value of the Hamiltonian despite the fact that it exists. These two problems are handled independently in the code.

The first problem, that of providing a Hamiltonian, is overcome by breaking with the standard tree model of having particles interact with tree nodes. As was discussed in the last section, this practice produces a system in which forces are not symmetric and the behavior cannot be described by Hamiltonian dynamics. Instead this algorithm adds terms to a Hamiltonian that represent the interactions of groupings of particles on the same level of the tree. As with a standard tree code, what groups are added as terms to the Hamiltonian depends on their size and the distance between them. Two leaves in the tree contribute to only one term in the Hamiltonian. That term will often represent the interaction of nodes higher up in the tree above those leaves. Figure 1 shows pictorially what terms might be added to the Hamiltonian in a 1D-tree structure.

To maintain the same Hamiltonian throughout the integration the structure of the tree is static. In our test code this was implemented by making the tree complete to a certain depth. An alternate method could be implemented that would allow for an adaptive tree. However, that would add complexity and it is not clear at this point if it would benefit most applications. By making the tree structure completely static, the program is also able to determine at start-up what pairs of nodes need to be considered in the Hamiltonian. This fact is useful for analytic treatments and demonstrations that the code is symplectic, but evaluating such a list is inefficient as in any realistic tree, many of the nodes, especially the leaves, are empty. Instead we say that a list defines the Hamiltonian, but recurse through the tree to do the calculations.

Even with this tree-generated Hamiltonian, the integrator would not be symplectic due to the fact that the time evolution of the Hamiltonian includes discontinuities as particles move between nodes in the tree. When particles move between nodes, the interactions they participate in change abruptly. This causes the values of specific terms in the Hamiltonian to jump in a discontinuous way. The correction to this problem involves the use of smoothing functions so that particles pass smoothly from one tree node to another. The use of smoothing functions in symplectic integrators is not new [9], however we are not aware of its use in the manner presented here.

The smoothing functions used here cause the mass of a particle to be “split” between nodes when it is near a node boundary. This produces the desired result that it removes discontinuities and allows the integrator to preserve the Hamiltonian. It also complicates matters in two ways. First, it causes the particle masses used in the terms of the Hamiltonian to be functions of position which complicates the equations of motion. Second, it increases the number of terms in the Hamiltonian in which each particle must be considered.

The smoothing functions themselves are quite simple. As with previous applications of smoothing functions, they need to have continuous derivatives to as high order as possible. In the case of the smoothing functions on a tree, we can use functions of the form

$$f_i(x, y) = g_i \left( \frac{x}{s} \right) \times g_i \left( \frac{y}{s} \right)$$

(6)

where s is the smoothing length, the subscript i denotes what node is being considered, and x and y are measured relative to the edge of the cell. While this smoothing function is independent of the z direction, all arguments can be extended to include z dependence. The function $g(x)$ need only satisfy
the following conditions. First, to ensure continuity \( g(x) = 1.0 \) for \( x \leq 1.0 \) and \( g(x) = 0.0 \) for \( x \geq 1.0 \). The derivatives of \( g \) should be zero at \(-1.0\) and \(1.0\) to at least the third derivative if not higher. Second, to ensure that particles experience the proper forcing when they are in the smoothed region between two nodes, we add the condition that \( g(x) + g(-x) = 1.0 \). Notice that this second requirement implies that with the exception of a constant term with value 0.5 a polynomial expansion will only include odd powers. This makes it fairly easy to generate smoothing functions that satisfy the condition of zero derivatives at the edges to high order.

A detail of the algorithm that should be discussed in depth is how to decide which pairs of nodes go in the Hamiltonian list. This is done in roughly the same way as the decision is made in a tree code to determine if a particle interacts with a node, or if it should go down to the children of that node to check for interactions. Unlike that decision, this one is made between two nodes at the same level of the tree. For two nodes to contribute to the Hamiltonian, the distance between them must be greater than some factor, \( \theta \), times the size of the nodes or they must be leaf nodes. In addition, none of the parents of the nodes should satisfy this condition. The simplest way to construct such a list is using a recursive function that recurses on two nodes in such a way that its arguments are always on the same level of the tree and the recursion terminates any time two nodes are added to the list. This style of recursive function is different from that seen in normal tree codes and is conceptually more complex as people are not used to recursion through data structures going through two paths at once. The pseudocode below shows how this can be implemented.

```
RecursiveInteractCheck(Node n1, Node n2, real theta)
  if (n1 is empty) or (n2 is empty) then return
  if (n1 is a leaf) or (distance(n1, n2) > theta * (size of n1)) then
    n1 and n2 interact
  else
    loop c1 over children of n1
    loop c2 over children of n2
    RecursiveInteractCheck(c1, c2, theta)
  end
```

Note that this recursive function has a very high branching factor. That is the reason for the first check. Our experience with an implementation of this algorithm showed that when the tree was deep, there were typically a large number of empty nodes that didn’t need to be evaluated. This is the reason one doesn’t actually keep a list of interacting nodes in the code because recursing through the tree is more efficient at pruning.

Each step in the integration requires placing particles in nodes on the tree, recursing the tree to calculate the derivatives to the Hamiltonian, and advancing the particles using a safe method. Our test code uses a straightforward second order T+V method for advancing the particles though the calculated derivatives could also be used with another form of symplectic integrator such as an MVS method [8] or the SYMBA method [9].

In the test code, the process of adding particles to the tree is accomplished by a recursive call that descends the tree structure as long as \( f(x,y) > 0 \). At each node that the particle falls in, the value of the smoothing function and its derivatives in the \( x \) and \( y \) directions are calculated and a structure referencing the particle and these values is added to the list of particles in that particular node. The values of the smoothing function and its derivatives are calculated and stored at this point because they are used multiple times during the calculation of the derivatives of the Hamiltonian.

After all the particles have been added to the tree a final pass is made to calculate relevant values for each node. In our test case we only used the total mass of particles in each node and their center of mass location. Once this has been done, the interaction list can be walked to calculate the derivatives of all the terms in the Hamiltonian. Each pair of nodes in the interaction list represents either the potential due to two interacting distributions of mass or to the sum of individual particle interactions. The later occurs when the nodes are leaves in the tree, while the former happens in all other cases. The interaction of leaves \( i \) and \( j \) when \( i \neq j \) produces the straightforward term

\[
H_{i,j} = - \sum_{k=1}^{n} \frac{G \times f_i(\vec{r}_k) m_i \times f_j(\vec{r}_k) m_j}{|\vec{r}_i - \vec{r}_j|}, \tag{7}
\]

where \( m_k \) is the mass of the \( k \)th particle and \( \vec{r}_k \) is its vector position. If \( i=j \) then the inner sum is taken from \( k+1 \) to \( n \). For any particles that are not in the smoothed region of a particular node, the value of the \( f \) function will be zero. This will be the case for most particles. For this reason, in practice these sums are taken by walking the list of particles in a node and therefore do not actually check the value of the \( f \) functions for particles from 1 to \( n \). When the nodes are not leaves, the term produced depends on the order of the mass expansion used for the nodes. We present here the first order case of treating the distributions as point masses located at the center of mass. In that case the term in the surrogate Hamiltonian has the form

\[
H_{i,j} = - \frac{G \times M_i \times M_j}{|\vec{R}_i - \vec{R}_j|}, \tag{8}
\]

where \( M_i \) and \( R_i \) represent the total mass and center of mass location of the distribution in the \( i \)th node. These have values given by

\[
M_i = \sum_{k=1}^{n} m_k f_i(\vec{r}_k), \tag{9}
\]

and

\[
\vec{R}_i = \frac{1}{M_i} \sum_{k=1}^{n} \vec{r}_k m_k f_i(\vec{r}_k). \tag{10}
\]

To update the particles we need the partial differentials of these terms with respect to the locations of the particles that are involved in them.

Taking the derivatives of (7) with respect to the position of the \( k \)th particle is straightforward. Without loss of
general we can assume that $k$ is a particle in node $i$. The derivative of (7) with respect to the $x$ position of the $k$th particle is given by

$$\frac{\partial H_{i,j}}{\partial x_k} = -\sum_{j=1}^{\kappa} G m_k m_j$$

$$r_{kl} f_j(|\mathbf{r}_k - \mathbf{r}_l|) = f_j(|\mathbf{r}_k|) f_j(|\mathbf{r}_l|) \frac{x_k - x_l}{r_{kl}},$$

where $r_{kl}$ is used to denote the value $|\mathbf{r}_k - \mathbf{r}_l|$. An analogous expression is found for the derivative with respect to $y$, while the $z$ derivative is simpler if the $f$ function is independent of $z$. The derivative of (8) is a bit more complex. Again we will look at the derivative with respect to $x_k$ where the $4$th particle is in the $4$th node. This produces the value

$$\frac{\partial H_{1,4}}{\partial x_k} = -G \frac{R_1 \left( M_j \frac{\partial M_j}{\partial x_k} - M_j M_k \frac{\partial R_y}{\partial x_k} \right)}{R_y^2},$$

where $R_y$ denotes $|\mathbf{R}_y - \mathbf{R}_y|$. To make this usable we need expressions for the two partial derivatives that appear in it. The derivative of the mass given by (9) is non-zero because as particles move through the smoothed regions at node boundaries the amount of mass they contribute to a particular node varies. The result of the derivative is

$$\frac{\partial M_j}{\partial x_k} = m_k \frac{\partial f_j(|\mathbf{r}_k|)}{\partial x_k}.$$  

The partial derivative of the distance between the nodes is more complex. It should be noted that (8) will never be used when $i$ and $j$ are adjacent nodes. In that situation (7) will always appear in the Hamiltonian instead. Combining that with the fact that the smoothing length should always be smaller than the node size gives us the result that the set of particles in nodes $i$ and $j$ will always be disjoint. With this in mind we can safely write the derivative as

$$\frac{\partial R_y}{\partial x_k} = \frac{1}{R_y} \left( \frac{\partial}{\partial x_k} \right) \left( \frac{\partial R_y}{\partial x_k} \right).$$

Last, we need to calculate the value of the derivative of the center of mass position with respect to $x_k$. This is equal to

$$\frac{\partial \mathbf{R}}{\partial x_k} = \frac{m_k}{M_j} \left( \hat{i} f_j(|\mathbf{r}_k|) + \frac{\partial f_j(|\mathbf{r}_k|)}{\partial x_k} |\mathbf{r}_k - \mathbf{r}_l| \right),$$

where $\hat{i}$ is the unit vector in the $x$ direction. In the case of the derivative with respect to $y$, this would be replaced by the unit vector in the $y$ direction. These expressions simplify for the $z$ derivative once again due to the fact that $f$ is not a function of $z$.

5 Results

Now that the method has been described, we can look at some results from applying it. A system modeling our own solar system with the Sun and four giant plants was used as a baseline for the simulations presented in this section. The tree based code was run on this system using several different parameters and compared to a standard integrator doing the full O(N^2) force calculation. Both use a T+V integrator. All the simulations were doing using a time step of 3.65 days so that there are on the order of 1000 time steps taken for each orbital period of Jupiter, the dynamical time for this system.

Because these simulations are of a system that is dominated by a central mass, another advantage of this method is used for the runs using the tree. The Sun is treated independently of the other particles and is not placed on the tree. So the terms involving the Sun in the Hamiltonian are always done on a particle-particle basis.

The first round of simulations were run for 1 million years to determine if the method conserves energy and preserves the expected behaviors of the system. Figure 2 shows the energy from three different simulations as a function of time. Note the scale on the vertical axis. This shows that the energy is conserved to better than one part in $10^4$ overall and is remarkably stable over the million year integrations. In all cases we see oscillations about a particular value with virtually no drift. The three data sets that are shown are for a simulation using the standard method, a simulation using the tree with a fixed smoothing width ($\sigma$ in equation 6), and a simulation using a variable smoothing width that scaled as the circular orbit velocity. Both of the tree simulations use $\theta=0.3$ and the tree was extended to a height of 4. This results in far more nodes than particles to insure that the test exercises both the motion of particles through the tree and interactions occurring at nodes above the leaf nodes.

![Figure 2: This plot shows the energy as a function of time for three different simulations. The top plot shows the standard method while the others show two variations on the tree method. The energy is in odd units, but note that the variations in energy are orders of magnitude smaller than the average value and the average is flat for the million year integration.](image-url)
For the tree methods one can also look at the value of the Hamiltonian calculated from the tree structure to see that it is preserved. Such a plot looks identical to the behavior of the energies in Fig 2 and is not shown separately.

Figure 3 shows the eccentricities of the four planets over time for the same three simulations. There is a well known periodic transfer of eccentricity from Jupiter the Saturn and back that is shown clearly in these plots [19]. The primary aspect to note is that this is preserved using the tree method as are the general trends in the orbits of Uranus and Neptune. The fact that all of the details are not preserved is not surprising as this system is chaotic and any variation in the numerical method will lead to variations in the detailed behavior of the system. The preservation of the gross behaviors is what is significant.

In order to test the speed of the tree method, varying numbers of smaller particles were added to the system of the Sun and four giant planets. This system was integrated for ten years. The length of time required to perform the integrations was measured with the Linux time command. Figure 4 shows the results for the standard scheme with squares, as well as the tree based scheme using different parameters for the tree. The circles are for $\theta=0.3$ and the triangles are for $\theta=0.1$. The size of the symbols indicates roughly the ratio of leaf nodes to particles that was imposed on the tree. The larger symbols went to a depth where there would be roughly 100 leaf nodes per particle while the smaller symbols had only 10 leaf nodes per particle. Since the number of leaf nodes is always a power of four, the level was chosen that would be closest to this multiple without going under. This resulted in trees with heights ranging from five to eleven, depending on the number of particles and the ratio.

Figure 4 shows that at small particle counts, the standard method is more efficient because it lacks the maintenance of the tree. However, between 500 and 2000 particles, depending on the parameters used, the time required using the tree method drops below that of the standard method. This is even true when the rather small $\theta$ value of 0.1 is used.

Power-law fits were made to the different timing results. As one would expect, the standard method is well fit by a power-law with a slope of 2. Using a power-law fit on the timings of the tree method gives a slope of 1.3, significantly better than the standard method. Of course, the goal of tree methods is typically to achieve $O(N \log N)$ performance. The fit using that function is inferior to that of the power-law, but larger particle counts would be required to determine whether or not this method is providing $O(N \log N)$ performance. Even the $O(N^{1.3})$ performance is far superior to the $O(N^2)$ performance given by the standard method and provides improvements in speed greater than an order of magnitude when more than 10,000 particles are considered.

6 Conclusions and Future Work

The basic results presented here are promising in that they clearly demonstrate that the tree-based Hamiltonian can be used with a standard symplectic map to integrate a system
with a speed that scales better than O(N^2). The method also appears to conserve this Hamiltonian and the true energy of the system over a million year timescale.

There are three main challenges with this method as it is currently implemented that we are looking at means of correcting.

- Speed of evaluation
- Numerical accuracy
- Accuracy of the Hamiltonian

The speed problem exists because each node in the tree contains a list of the particles under it, and when we calculate a term of the Hamiltonian involving a node with another node, we must run through that list for both nodes. The numerical accuracy problem is due to the fact that the terms involving the derivative of the smoothing function can be much larger than those that don't. Because the terms involving the derivatives of the smoothing function typically come close to canceling out, this introduces numerical errors.

We are looking at a single solution for both of these problems. The general idea is to alter the method of evaluation on the tree so that it is done in two passes, but we don't have to run through all the particles for a node in each interaction. Instead, the nodes store particle independent values in them in one pass and the particle dependent values are pulled in during a second pass. This method will also allow us to separate the terms involving the smoothing function from those that don't and combine them more intelligently for each particle.

The third problem arises because our tree-based Hamiltonian is only using monopole moments for the mass distributions in a node. We have looked at going to quadrupole moments. This is more complex for node-node terms than for the node-particle terms considered in other tree codes and the math required was beyond the bounds of what could be put in the length of this conference paper. As the code stands, it does not behave well when θ is increased to 0.5. Using a higher order expansion of the mass below each node could fix this.

Future publications will explore our solutions to these three challenges as well as look more closely at the method in regards to speed and conservation of both the tree-based Hamiltonian and the true energy of the system over extended periods of time.

Acknowledgments
This project was supported by an internal, quick-look grant from the Southwest Research Institute. All plots for this paper were made using SwiftVis, developed with support from NASA AISRP.

7 References