AAS 10-225

EFFICIENT PARALLELIZATION OF NONLINEAR PERTURBATION ALGORITHMS FOR ORBIT PREDICTION WITH APPLICATIONS TO ASTEROID DEFLECTION AND FRAGMENTATION

Brian D Kaplinger,[*] Bong Wie[†] and David Dearborn[‡]

Orbit determination and control problems are sometimes solved using linearized methods, even though nonlinear effects can sometimes dominate the system. This paper addresses ways to parallelize computational algorithms for numerical integrators and nonlinear relative motion equations. An application of the proposed methods to mutual gravitation of a fragmented asteroid is presented, with analysis of the relative efficiency of parallel architechtures. Hydrodynamic simulation of a subsurface nuclear explosion is used to generate the initial conditions, after which the fragmented system is propagated in parallel including mutual gravitational terms. Efficient algorithms for this problem emphasize the achievement of verifiable results for asteroid deflection/fragmentation research using limited or budget-allocated computer resources.

INTRODUCTION

Astrodynamical problems involving orbit determination and prediction are typically solved by assuming that orbital perturbations are locally linear to take advantage of the efficient methods available for processing these calculations. A complication arises when the problem addressed is perturbed from a simple 2-body problem solution, whether by the presence of additional gravitating bodies or an external force. Computational simulation of these nonlinear problems can be difficult and expensive. This paper addresses ways to parallelize perturbation algorithms for relative motion problems with an elliptical nominal reference orbit. The example presented in this paper resulted in quasi-linear speedup with low level parallel implementation. Splitting calculations up at the algorithmic level allows modern multi-core desktop systems to tackle problems previously relegated to clusters of high-performance computers. The application problem addresses an asteroid fragmented during an attempt to deflect it from an impact trajectory. Hydrodynamic simulation of a nuclear explosion below the surface of the asteroid was used as an initial condition to generate a fragmented body moving at realistic dispersion speeds. The result shows the ability of the proposed computational model to handle several interacting bodies, as well as emphasizing the efficient use of orbital simulation algorithms to achieve cutting-edge and verifiable results in the planetary defense community. A major barrier to the progress of asteroid fragmentation research at the university level has been the bottleneck in computational resources available to these groups. This paper will show

[*] Graduate Assistant, Asteroid Deflection Research Center, Iowa State University, 2271 Howe Hall, Room 2355, Ames, IA, 50011-2271, bdkaplin@iastate.edu.

[†] Vance Coffman Endowed Chair Professor, Asteroid Deflection Research Center, Iowa State University, 2271 Howe Hall, Room 2355, Ames, IA, 50011-2271, bongwie@iastate.edu.

[‡] Research physicist/astrophysicist, , Lawrence Livermore National Laboratory, 7000 East Avenue, Livermore, CA, 94550, dearborn2@llnl.gov.

quick ways to allow for perturbation methods to be parallelized and applied on both Shared-Memory Parallel (SMP) and Distributed-Memory Parallel (DMP) architectures. This brings the solution of complex astrodynamical problems within the range of desktop workstations or small groups of computers clustered using a local area network. The issues of load balancing will be discussed, and the resulting improvement in time-to-solution is covered to emphasize the strengths and weakness of the parallelization methods available to researchers.

PARALLELIZATION

Finding a motivation for parallelization of a particular algorithm is usually not a difficult task. Even basic computations repeated for design cycle purposes, optimization, or other forms of iterative improvement, can explode the required computing time to unacceptable levels. The case of the application problem presented later is an example of where analysis of input conditions requires repeated runs of the model simulation. With millions of required calculations, short time steps or repeated runs become prohibitive in terms of computational cost. Many researchers believe that parallelizing their algorithms for compiled simulations is only necessary if they plan to run them on a supercomputer. However, modern multi-core desktop systems are readily available in most research environments, and proper use of parallelization techniques can reduce time-to-solution for simulation programs. Two main types of parallel platforms are available, and they have corresponding techniques and libraries for their use. Two open source libraries, OpenMP and Message Passing Interface (MPI), are discussed in this paper. The reader is provided with examples of relevant grammar and library references needed, and is referred to other sources for further study.

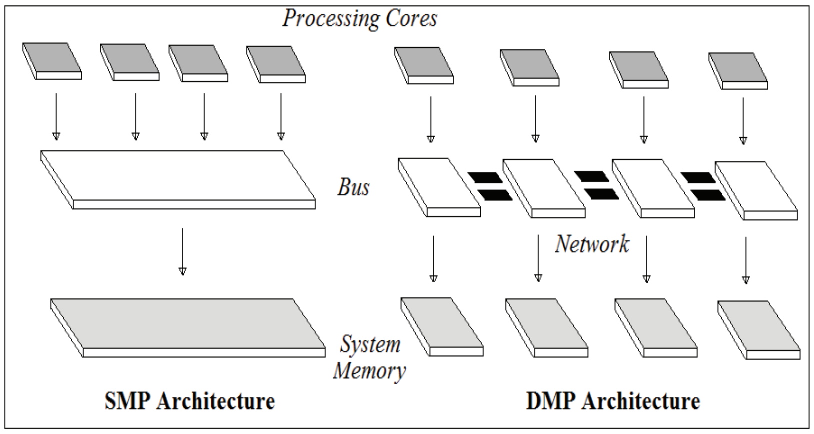

Figure 1. Comparison of SMP and DMP Architecture

Description of Parallel Architectures

As previously mentioned, there are two main types of parallel computer systems. The first is called Shared Memory Parallel (SMP) Architecture, while the second is called Distributed Memory Parallel (DMP) Architecture. Each computer system, often identified as its printed circuit board (also known as a system board, main board, or motherboard), will be referred to as a node.[1] Each Central Processing Unit (CPU) is represented virtually as the capability to run a single program (or thread), and will be referred to as a core. In the case where a system has one node, but many cores, the cores can all access the same memory bank. This type of computer, common of multi-core desktops, is a standalone SMP architecture. DMP systems will often have several nodes, each with its own memory bank that is unreadable by the other nodes. Communication that is required between nodes is handled using networking technology or other communication equipment.[1,2] Figure 1 shows a visual comparison of SMP and DMP architectures. Traditional DMP parallelization can have more cores than physical nodes, but are still treated as if each core is a node. Multi-core systems that are networked together are handled by running threads on each core independently. Composite approaches utilize one library for communications between physical nodes while treating each node as a SMP system. These approaches can be more complicated and prone to programming errors, but offer benefits in memory usage and efficiency. Figure 2 shows an example composite architecture, divided into SMP and DMP regions. More complicated architectures are also used, though combinations of these two are most likely to be encountered by the average researcher.

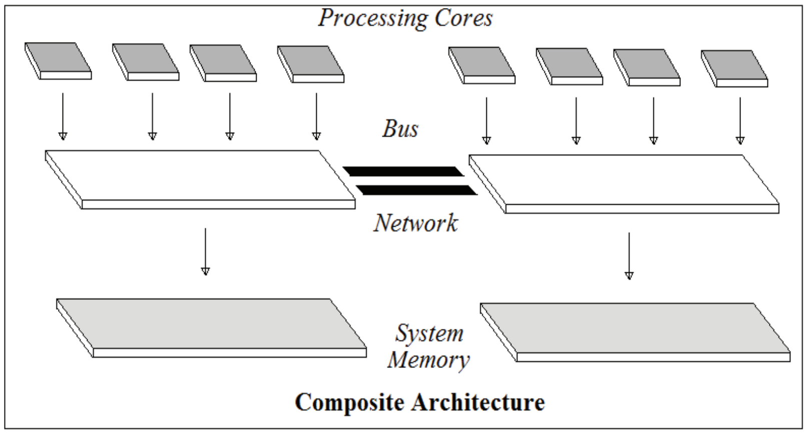

Figure 2. Example of a Composite SMP/DMP Architecture

Introductory OpenMP

OpenMP is an Application Program Interface (API) that facilitates parallelism in SMP environments. It consists of a library of architecture-dependent variables, functions, and subroutines that are useful in speeding up the processing of algorithms. It also consists of compiler directives through which the programmer can direct the creation of threads, the use of memory, and other application-specific needs.[3] OpenMP specifications exist primarily for C, C++, and Fortran programs. They extend the basic language specifications to add loops, constructs, and communication necessary to fully utilize an SMP system, while maintaining the portability of using a compiled language for simulation. Programming directives are given as comments, so compiling them on a system without OpenMP results in the creation of a serial program without changes being made to the source code. Specific examples will be given in the Fortran language, though equivalent translations in C and C++ are available in the OpenMP specification guide.[3]

The use of OpenMP gets started at the beginning of a program by including the OpenMP library header in the source:

```
INCLUDE "omp_lib.h"
```

A sample (albeit trivial) use of OpenMP directives in adding vectors is shown below:

```
!$OMP PARALLEL DO
    DO i = 1,n
        z(i) = x(i) + y(i)
    END DO
  !$OMP END PARALLEL DO
```

In this example (known as a loop directive), the OpenMP directive is given as a comment opening up a parallel region. The loop interior to the directive is split up onto the available threads, which is most efficient when the total number of loops n is an integer multiple of the number of available threads. By default, the compiler should direct the executable to spawn a number of threads equal to the number of cores upon encountering the PARALLEL directive. Using a quad-core desktop as an example, four threads would be spawned numbered 0-3 (with 0 being the parent, or root, thread). Thread 0 would execute the previous loop for $i = 1, 5, 9, ...$, while thread 1 would execute the loop for $i = 2, 6, 10, ...$, etc. An alternative setting is to have thread 0 execute $i = 1, ..., n/4$, while thread 1 executes $i = n/4 + 1, ..., n/2$, etc. This method would take advantage of many types of array storage, in which values are stored next to adjacent values in the array.[1,3]

One key issue that should be brought up for SMP architectures is that in certain cases threads need to have private memory allocated to them. In the above array example, no harm is done by all threads having access to the x, y, and z arrays, because each thread is only altering the values of the array on the loop counter variable (i in this case) it has access to. The following code would produce an undesired error:

```
!$OMP PARALLEL
  a = 0
    DO i = 1,n
        a = a + i
    END DO
    WRITE(*,*) a
  !$OMP END PARALLEL
```

One might expect the code to produce the sum of the loop counter variables, but remember that the threads in the parallel region share the value of a. So, for each computation of a, a thread pulls the current value from memory, adds the current counter value to it, and replaces it in memory. All of the threads doing this simultaneously, and at potentially slightly different rates, would yield garbage as a result. A separate value of a could be written to the screen depending on which thread is finished, and they may not even match. A better way of conducting this calculation is illustrated below, in which a separate copy of the variable a is created in memory for each thread. When a thread changed the value of a, it is doing so only to its private view of the system memory, so there is no overlap to interfere with the computation. Note that a private value of the loop counter i must also be used since it appears in the calculation. This parallel region is not a loop construct, so each thread does a separate run through the loop rather than sharing it.

```
!$OMP PARALLEL PRIVATE(a,i)
  a = 0
    DO i = 1,n
```

```
        a = a + i
    END DO
    WRITE(*,*) a
  !$OMP END PARALLEL
```

Sometimes it is necessary for a thread to know its thread number in order for the programmer to more explicitly direct its actions. In that case, the `OMP_GET_THREAD_NUM` subroutine would be used. Several of the routines available in the OpenMP library, such as `OMP_GET_NUM_THREADS`, can be used within parallel regions to set and determine the local working environment. As has been shown, OpenMP works through directing areas of a serial program to spawn teams of computing threads at specific points. These teams carry out the directed task, and then are reintegrated back into a serial section of the program. This allows for actions typically reserved for serial codes (such as input and output) to be taken, and for parallel execution of specific tasks. It often allows existing serial code to be parallelized in a few step by identifying key loop and memory structures to be split up among threads. Parallel regions in a SMP program can be nested, though the exact application determines whether or not an error would occur. Further programming guidelines using OpenMP can be found in the API manual.[3]

Introductory MPI

Contrasted to OpenMP, traditional implementations of Message Passing Interface take place using an external process scheduler. A package including MPI and MPI-compliant compilers must be installed on the DMP architecture. Specific implementation, including communication among nodes, must be handled by the programmer. For this reason, the algorithms presented in this paper will be restricted to OpenMP, though the relevant commands for MPI implementation can be found in Reference 2. A basic introduction to MPI commands will be given here for the interested reader.

Since the spawn of processes is handled outside of the source code, the simulation needs to be programmed with the knowledge that an individual executable will be run by each computing thread. Linking the source library is similar to OpenMP, and one way to acheive this is by including the applicable MPI header file:

```
  INCLUDE "mpif.h"
```

Every MPI program must call the initialization routine `MPI_INIT`, which gathers the environment variables (such as `MPI_COMM_WORLD`, which establishes the communication environment) and makes them available to the executing thread. Once this has been done, library routines such as `MPI_COMM_RANK` and `MPI_COMM_SIZE` can be used to determine the rank of the executing process (rank 0 is the root or master rank) and the total number of processes running on the DMP system. A benefit of this approach is that physical nodes can be added or removed from the computing environment arbitrarily, with the external manager deciding how many processes are available to run an executable. When the parallel computations are completed, the program must call `MPI_FINALIZE` to close up running processes and pending communication between nodes.[2,4]

Two primary communication routines are needed for a naive implementation of MPI: `MPI_SEND`, and `MPI_RECV`. The send and receive routines function basically as one would expect them to, but require additional arguments to identify what communication is being sent and what route it should take. The sending rank and receiving rank must be known, as well as a tag to identify the message (often an integer). The type of variable being sent is needed to ensure proper translation of the message. Also useful to the programmer is the concept of a communicator, which is a communication subroutine (similar to `MPI_SEND`) that can be set aside specifically for a library routine. Sending this external routine as an argument to library routines avoids confusion with the desired communications programmed by the researcher. Additional information concerning communicators other than `MPI_SEND` and `MPI_RECV` can be found in Reference 4. When an instance of `MPI_SEND` with a specific message and tag matches an instance of `MPI_RECV`, the communication is handled by the networking equipment connecting the nodes. The following is an example of a parallel "Hello World" program using MPI, which can be found in Reference 4 - an excellent source for the interested reader.

```
call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, procs, ierr)
if (rank /= 0) then
    write(string,"(I3)") rank
    message = 'Greetings from process '//trim(string)// ' !'
    dest = 0
    tag = 0
    call MPI_SEND(message, message, MPI_CHARACTER, &
    & dest, tag, MPI_COMM_WORLD, ierr)
else
    do source = 1, procs-1
       tag = 0
       call MPI_RECV(message, 100, MPI_CHARACTER,
       & source, tag, MPI_COMM_WORLD, status, ierr)
       write(*,*) message
    enddo
endif
call MPI_Finalize(ierr)
```

ALGORITHMS

This section will present some common algorithms in orbit determination and prediction, starting with a trivial integration problem to demonstrate the concepts. In each algorithm, relevant points where parallelization can be applied are identified, and pseudo-code is shown to maintain brevity. It is assumed that the reader can program the computations necessary from the given equations.

Integration Examples

A basic example of a problem for which parallelization is simple is the use of the Trapezoid Rule to compute the integration of a function. The Trapezoid Rule gives an estimate for the value of an integral by adding up trapezoids from $x = a$ to $x = b$ under the area of the function:[5]

$$\int_a^b f(x)dx \approx \frac{b-a}{2n}(f(x_0) + 2f(x_1) + \cdots + 2f(x_{n-1}) + f(x_n)) \tag{1}$$

where x_i is the value of x at which the function is evaluated and the interval (a, b) is divided into n equal partitions. For concreteness, we will examine the parallel implementation of this rule for a given external function $f(x)$ and a known number of subdivisions n. We assume that an array of the independent variable $x(i)$ at which the function is to be evaluated has been provided.

```
     p = OMP_GET_NUM_PROCS()
     np = FLOOR(n/p)
     ALLOCATE(integral_part(p))
!$OMP PARALLEL PRIVATE(t, i)
   t = OMP_GET_THREAD_NUM()
   integral_part(t+1) = f(x(t*np+1))
   IF (t/= p-1) THEN
        DO i = t*np+2,(t+1)*np-1
           integral_part(t+1) = integral_part(t+1) + 2*f(x(i))
        END DO
        integral_part(t+1) = integral_part(t+1) + f((t+1)*np)
     ELSE
        DO i = t*np+1,n-1
           integral_part(t+1) = integral_part(t+1) + 2*f(x(i))
        END DO
        integral_part(t+1) = integral_part(t+1) + f(n)
     END IF
   !$OMP END PARALLEL
     integral = SUM(integral_part)
     WRITE(*,*) integral
```

Gradient Optimization

In trying to determine a particular orbit for simulation, researchers often implement shooting methods to solve for the orbital parameters of the trajectory meeting required boundary values. Since the system of possible orbits behaves nonlinearly, small changes in orbital elements can result in drastic changes to the position as a function of time. One common shooting method is Newton's method with the initial condition as the dependent variable. A basic implementation of Newton's method is given as:[5]

$$y_{j+1} = y_j - \frac{f(y_j)}{f'(y_j)} \tag{2}$$

where y_j is the current guess of a root to the function $f(y)$, and $f'(y)$ is the derivative of the function at that point. This can be implemented as a shooting method by taking $f(y)$ to be the difference between the desired state of the system at the boundary and the resulting state using the initial condition vector y. Taken this way, $f(y)$ does not often have an analytical derivative. This requires the derivative of $f(y)$ (partial derivatives for each dependent variable in the multivariable case) to be estimated. One way to do so is a finite difference formula using a small step size h:[5]

$$f'(y) \approx \frac{f(y+h) - f(y-h)}{2h} \tag{3}$$

This type of shooting method is illustrated below, in which the parallel implementation uses one thread per dependent variable. The partial derivatives of the system state with respect to the initial conditions are estimated, with each thread making only the function calls it needs to determine the required partial derivative by calculating $f(y+h)$ and $f(y-h)$. A limiter is placed on the change in estimate in order to avoid sign changes, and the optimizer is under-relaxed by multiplying $f(y)/f'(y)$ by the step size (a very small value). This helps to ensure continuity of the solution while mitigating the effects of unstable choices in input conditions. This process is repeated until the optimizer converges.

```
CALL opt_function(state0, f)
!$OMP PARALLEL DO PRIVATE(j, state, fL, fR, df)
DO i = 1,n
    ! rewrite current values of system state to most recent iteration
    DO j = 1,n
        state(j) = state0(j)
    ENDDO
    state(i) = state0(i)*(1.d0+h)
    CALL opt_function(state, fR)
    state(i) = state0(i)*(1.d0-h)
    CALL opt_function(state, fL)
    df = (fR-fL)/(2.d0*h*state0(i))

    ! update guess for initial conditions
    IF (ABS(f/df)<ABS(state0(i))) state0(i) = state0(i) - h*f/df
END DO
!$OMP PARALLEL DO
```

Equations of Relative Motion

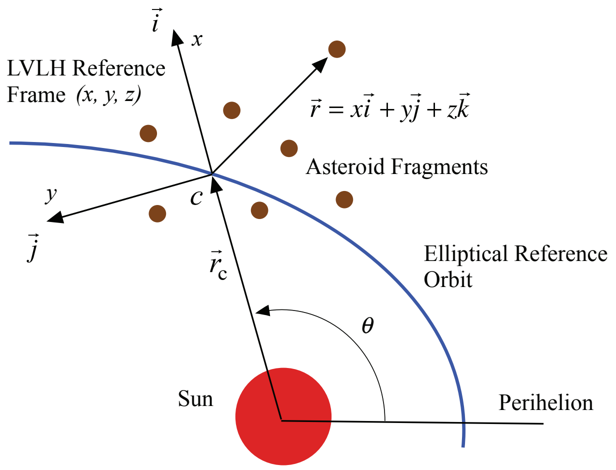
$\vec{i}$
x
LVLH Reference Frame (x, y, z)
$\vec{r} = x\vec{i} + y\vec{j} + z\vec{k}$
Asteroid Fragments
y
c
$\vec{j}$
$\vec{r}_c$
Elliptical Reference Orbit
θ
Sun
Perihelion

Figure 3. Rotating Local Vertical Local Horizontal (LVLH) Frame

Orbit prediction for rendezvous or proximity operations in an inertial reference frame can often be complicated. Geocentric or heliocentric distances are on the order of thousands to millions of kilometers, while satellites intending to rendezvous could be mere kilometers apart. For this reason, reference frames to determine relative motion are often introduced. One such reference frame is the Local Vertical Local Horizontal (LVLH) frame illustrated in Figure 3, which rotates along the nominal orbital trajectory. In this figure, r_c is the “chief” radius, or the distance from the gravitating body to the nominal orbit. The x coordinate measures radial distance, y measures transverse distance along the direction of rotation, and z measures the distance out of plane.

The general equations of relative motion in this coordinate frame are given as:[6]

$$\begin{aligned}\ddot{x} - 2\dot{\theta}\Big(\dot{y} - \frac{\dot{r}_c}{r_c}y\Big) - \Big(\dot{\theta}^2 + \frac{\mu}{r_c^3}\Big)x &= -\frac{\mu}{r_d^3}(r_c + x)\\ \ddot{y} + 2\dot{\theta}\Big(\dot{x} - \frac{\dot{r}_c}{r_c}x\Big) - \dot{\theta}^2 y &= -\frac{\mu}{r_d^3}y\\ \ddot{z} &= -\frac{\mu}{r_d^3}z\end{aligned} \tag{4}$$

where $r_d = \sqrt{(r_c + x)^2 + y^2 + z^2}$ is the “deputy” radius, or the distance from the gravitating body to an object in the LVLH frame. It should be noted that this system is distance preserving, so x, y, and z are physical distances and not determined by the nominal orbit radius r_c. A low level parallelization for this system is to calculate the acceleration for each body independently and integrate. This becomes especially more applicable as we add perturbing effects to the acceleration, which will be discussed later. For now, the following pseudo-code using a function call for accelerations would suffice for our purposes. Note that body-specific variables that are not dependent on our loop counter need to be private. If x, y, and z are arrays of dimension n, we have:

```
!$OMP PARALLEL DO PRIVATE(rd, ax, ay, az, xnew, ynew, znew)
    DO i = 1,n
        rd = SQRT((rc+x(i))**2 + y(i)**2 + z(i)**2)
        CALL accelerations(x(i), y(i), z(i), rc, theta, rd, ax, ay, az)
        CALL integrator(ax, ay, az, xnew, ynew, znew)
        x(i) = xnew
        y(i) = ynew
        z(i) = znew
    END DO
  !$OMP END PARALLEL DO
```

APPLICATION PROBLEM

There is evidence that the impact of a near-Earth object (NEO) would constitute a significant threat to life on Earth. Deflection methods of sufficiently high energy density are required, and one proposed method is the use of nuclear explosive devices above, on, or beneath the surface of an NEO.[7] The level of energy imparted to the NEO from these methods makes fragmentation of the target a plausible outcome.[8,9] In some cases, thermal ablation of surface material causes compression waves to propagate through the remainder of the NEO. The stress of these waves may be enough to continue fracturing the material.[10] A simulated surface explosion on an NEO was used to generate positions and velocities of a fragmented body due to a deflection attempt. While lead time has often been suggested as the most important variable affecting asteroid deflection, it has been shown that deflection 15 days ahead of impact is sufficient to reduce the total mass of the asteroid impacting Earth.[11] This can be beneficial in situations where some impacting mass is inevitable, and may reduce the size of impacting bodies to maximize the ablative protection offered by Earth’s atmosphere.[9,11]

Input Formulation

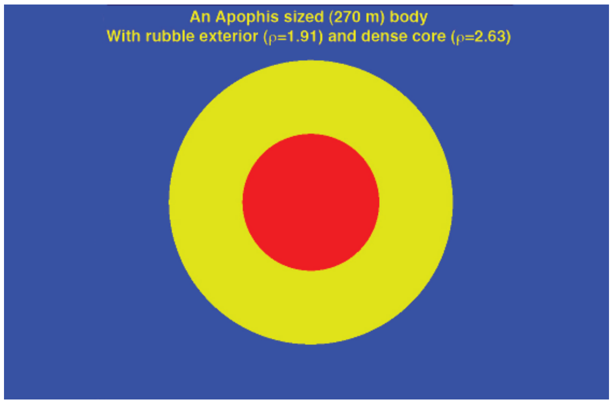
An Apophis sized (270 m) body
With rubble exterior (ρ=1.91) and dense core (ρ=2.63)

Figure 4. Structural Model of an Apophis-sized (270 m) Body

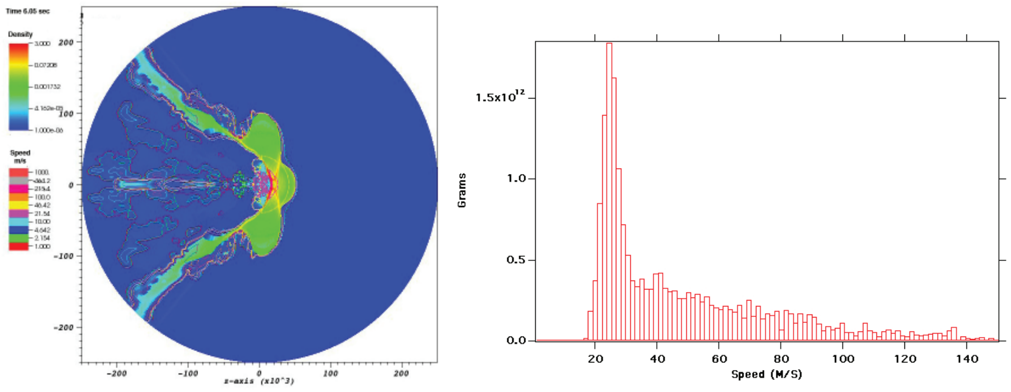

Figure 5. Distribution of Fragments and Velocities for 300 Kt Subsurface Explosion

The input conditions for the simulation shown are identical to those presented for the smaller case in Reference 11, where they can also be found. A two component spherical structure with a high density core consistent with granite and a lower density mantle was constructed to represent soft, porous rock formed by the compaction of volcanic ash. The bulk density of the structure was 1.99 g/cm^3, close to that measured for asteroid Itokawa.[12] The model was sized to approximate the asteroid Apophis with a diameter of 270 meters, and a total mass of 2.058×10^{13} g, or a bit over 20 million tons, as shown in Figure 4. A subsurface explosion was simulated on the body by sourcing in energy corresponding to 300 Kt to a cylindrical source region 1 m in diameter and 5 m long. The energy source region expands, creating a shock that propagates through the body resulting in fragmentation and dispersal. The structure of the asteroid was modeled with a linear strength model and a core yield strength of 14.6 MPa. The mass-averaged speed of the fragments after 6 seconds was near 50 m/s with peak near 30 m/s, as shown in Figure 5. A three-dimensional fragment distribution was constructed from the hydrodynamics model by rotating the position, speed, and mass of each zone to a randomly assigned azimuth about the axis of symmetry. Figure 5 also shows the two-dimensional distribution of body fragments after the completion of the subsurface explosion simulation.

An impacting trajectory was designed using the gradient approach to Newton's method discussed previously. J2000 data was used to model Earth's orbit as described in Reference 13. Orbital elements for an impacting body were chosen using a shooting method. The orbital elements of the nominal (heliocentric) trajectory at the time of Minimum Orbit Intersection Distance (t_{moid}) are shown in Table 1. The axis of symmetry on which the explosion occurs was aligned with the transverse direction of the asteroid orbit, and the dispersion of the fragments was then propagated.

Table 1. Orbital Elements of Impact Scenario for t_{moid} = 2454834.5 JD

Orbital Parameters	Value
Semimajor Axis, a (AU)	2.1037
Eccentricity, e	0.5377
Inclination, i (deg)	11.2738
Longitude of Right Ascension, Ω (deg)	282.5990
Argument of Perihelion, ω (deg)	194.3564
Mean Anomaly at t_{moid}, M_0 (deg)	351.4929

Integrators and Error

A fourth-order Runge-Kutta integrator was used to integrate the general equations of relative motion with respect to the nominal asteroid trajectory, discussed earlier. The output of these equations was compared to the elliptical Clohessy-Wiltshire-Hill (CWH) equations. These equations are a linearized version, while the model presented uses a two-body problem solution with perturbation terms, calculating the relative distance from the nominal orbit for each fragment. A global time step of 1 hour was chosen for computational efficiency, with an adaptive error-limiter that ensures smooth transition at each time step. The surface explosion is modeled as 15 days prior to impact, and the orbital dispersion of the resulting cloud of fragments was calculated.

The elliptic CWH equations used for comparison are (in the same LVLH frame as discussed earlier):[6]

$$\begin{aligned} \ddot{x} - \left(\dot{\theta}^2 + 2\frac{\mu}{r_c^3}\right)x - \ddot{\theta}y - 2\dot{\theta}\dot{y} &= 0 \\ \ddot{y} - \left(\dot{\theta}^2 - \frac{\mu}{r_c^3}\right)y + \ddot{\theta}x + 2\dot{\theta}\dot{x} &= 0 \\ \ddot{z} + \frac{\mu}{r_c^3}z &= 0 \end{aligned} \tag{5}$$

where r_c is the nominal distance of the system center of mass to the sun, and θ is the true anomaly of the nominal orbit. These can be directly integrated from the initial conditions using the following equations:[6]

$$\ddot{r}_c - \dot{\theta}^2 r_c + \frac{\mu}{r_c^2} = 0 \tag{6}$$

$$\ddot{\theta} + \frac{2\dot{r}_c\dot{\theta}}{r_c} = 0 \tag{7}$$

Earth's Gravity Field

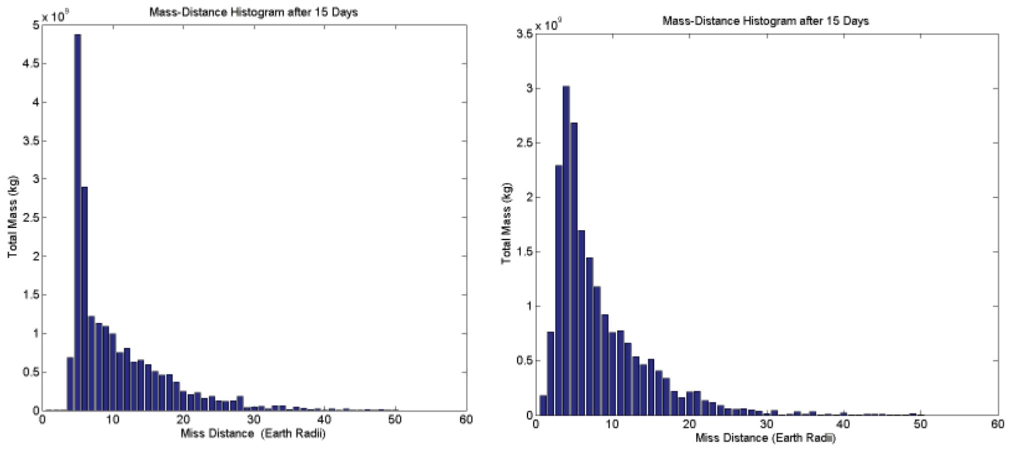

Figure 6. Comparison of Fragment Miss Distance for Keplerian Orbits in Static and Dynamic Models

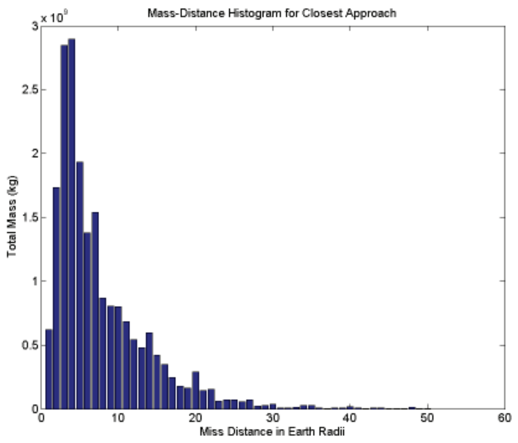

Figure 7. Miss Distance for Model Including Earth Gravity

Figure 6 shows the distance of closest approach to the Earth for fragments using only Keplerian orbits for each fragment. The left figure shows the miss distance of each fragment as measured at the time of expected impact (static case). While it would appear that after 15 days dispersion along the asteroid orbit results in no impacting mass, the results of a dynamic simulation shown by the right figure indicate otherwise. When the Earth is modeled as passing through the cloud of fragments over time, there is a small amount of expected impacting mass. The large amount of fragments passing very close to the Earth but not impacting would suggest that some trajectories are altered by the Earth's gravity. If we transform the heliocentric coordinates of the Earth's orbit into coordinates relative to the nominal asteroid orbit (x_E, y_E, z_E), we can then add a gravitational perturbation corresponding to the Earth to the equations of motion for each fragment. This perturbation increases the impact mass to around 3% of the original asteroid mass, as shown in Figure 7.

Mutual Gravitation

Including the mutual gravitational terms between the fragments and the effect of Earth's gravity, the final equations of motion for each fragment are below, where μ_E is the gravitational parameter of the Earth, r_{Ei} is the distance from fragment i to the Earth, G is the universal gravitational constant, and m_j is the mass of fragment j:

$$\begin{aligned}
r_{Ei} &= \sqrt{(x_E - x_i)^2 + (y_E - y_i)^2 + (z_E - z_i)^2} \\
r_{ij} &= \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2} \\
\ddot{x}_i &= 2\dot{\theta}\Big(\dot{y}_i - \frac{\dot{r}_c}{r_c}y_i\Big) + \Big(\dot{\theta}^2 + \frac{\mu}{r_c^3}\Big)x_i - \frac{\mu}{r_d^3}(r_c + x_i) + \frac{\mu_E(x_E - x_i)}{r_{Ei}^3} + \sum_{j\neq i}\frac{Gm_j(x_j - x_i)}{r_{ij}^3} \\
\ddot{y}_i &= -2\dot{\theta}\Big(\dot{x}_i + \frac{\dot{r}_c}{r_c}x_i\Big) + \Big(\dot{\theta}^2 - \frac{\mu}{r_d^3}\Big)y_i + \frac{\mu_E(y_E - y_i)}{r_{Ei}^3} + \sum_{j\neq i}\frac{Gm_j(y_j - y_i)}{r_{ij}^3} \\
\ddot{z}_i &= -\frac{\mu}{r_d^3}z_i + \frac{\mu_E(z_E - z_i)}{r_{Ei}^3} + \sum_{j\neq i}\frac{Gm_j(z_j - z_i)}{r_{ij}^3}
\end{aligned} \tag{8}$$

It should be noted that this model is collisionless and does not account for reconstitution of fragments. In order to limit the acceleration provided, the relative distance r_{ij} was limited to the contact distance between spherical bodies with a constant density $\rho = 1.99$ g/cm^3. Therefore, the fragments are allowed no closer to one another than if their surfaces were physically in contact.

Results and Discussion

Figure 8 shows the relative increase in impacting mass due to mutual gravitational terms among the fragments. A total impacting mass of around 7% is observed, more than twice the simulation result without mutual gravitational effects. The cloud of debris is concentrated closer to the nominal center of mass, as expected, resulting in closer passes by the Earth. One might expect that lower average dispersion velocities make the effect of gravity more predominant, and in fact that is the case. Figure 9 plots the ratio of impacting mass in the mutual gravity case compared to the non-gravity case as a function of initial dispersion speed. Examination of the trend shows that with scaled down initial velocity from the explosion, the effect of gravity becomes more predominant and results in a larger displacement compared to the non-gravitational case.

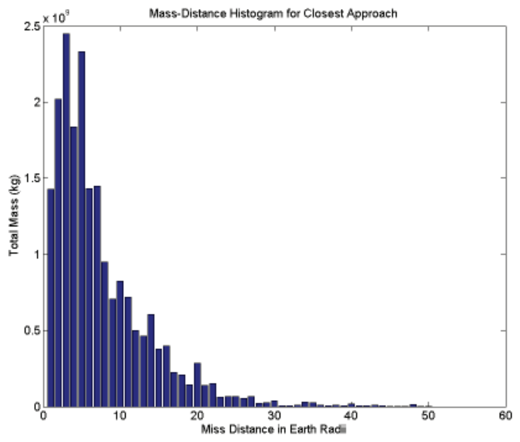

Figure 8. Fragment Miss Distance in Earth Radii (Mutual Gravitation)

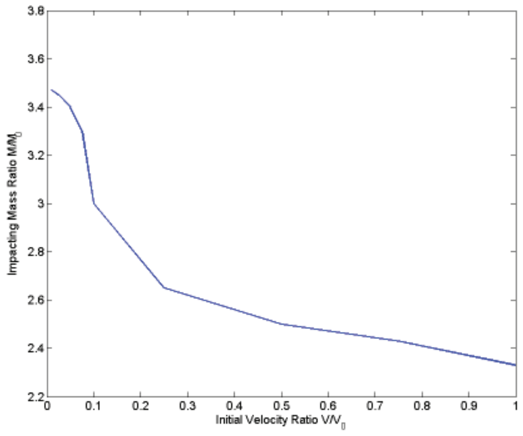

Figure 9. Effect of Initital Velocity on Relative Importance of Mutual Gravitation

COMPUTATIONAL ISSUES

The problem of a cloud of fragments with mutual gravitation is one that emphasizes the problems with brute force computation for engineering simulation. Large numbers of fragments result in substantially larger computation times, and in fact the cost of many simulations expands nonlinearly with the number of variables. Some algorithms have computational needs proportional to the square (or even the cube) of the dimensionality of the problem.

Scaling and Repetition

The application problem presented is an example where expanding the dimensionality of the problem quickly leads a researcher into trouble. Compared to past research done with 5,000 fragments, the current paper uses input conditions including over 18,000 fragments. Having more than three times the number of fragments results in over 9 times the number of necessary calculations per time step. Expanding input conditions to accommodate an available model with 200,000 fragments would multiply the current required computer time by over 100 times. It becomes apparent that engineering simulation can be sensitive to scaling, and that splitting up the burden on a parallel system is desirable. Also, many problems in engineering design cycles or optimization require the testing of several input variables to maximize (or minimize) the output. The same is true of shooting methods for boundary value problems, especially those employing finite element methods. These problems are best broken up into independent groups of input variables and distributed on a parallel system, which can result in almost linear improvement in computational time.

Balancing and Example Problem Efficiency

One issue that is always of concern to the parallel programmer is balance. If several threads are set to execute a set of code simultaneously, but some of them have less work to do (in terms of required computing time), then those threads are forced to wait on the completion of the work by the other threads. Therefore, it is always in the programmer's best interest to evenly distribute the work so that CPU time is not wasted waiting on communication. In particular, when a thread reaches a communication point we want all of the other threads to arrive at that point at approximately the same time in order to minimize the latency across our network (for DMP systems) or across our memory bus (for SMP systems).

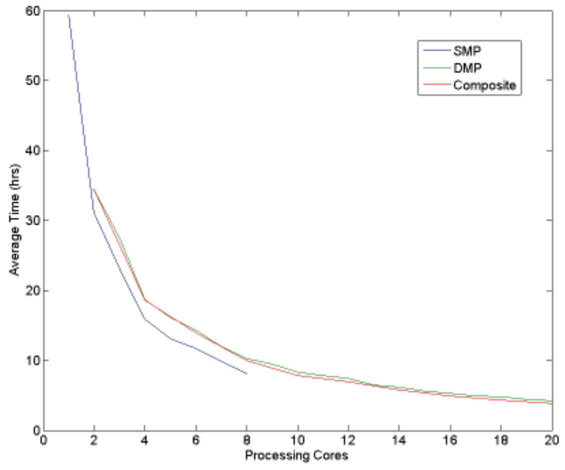

Figure 10. Total Elapsed Time for Fragmentation Problem

Figure 10 shows the reduction of computational time required for the example fragmentation problem when parallelized. It can be seen that with low level SMP parallelization of the gravitational perturbation terms, additional cores result in almost linear return of computational time. For systems with more than four cores, two types of parallelization are shown. A basic DMP architecture (or cluster of single cores) has a significantly higher cost than an equivalent SMP system, but is substantially more capable of scaling to higher numbers of cores. A composite programming structure is also shown (in this case grouping nodes into SMP regions of 4 cores each). Equivalent implementations with 2 cores and 8 cores per node are also effective, though not shown for this problem.

CONCLUSION

This paper addressed ways to parallelize nonlinear relative motion equations for use in orbit determination and prediction. Input conditions for an orbital dispersion model of a fragmented asteroid were presented, with the effect of Earth's gravitational field and mutual gravitational perturbations between the fragments. After 15 days of lead time, a subsurface explosion of 300 Kt nuclear explosive for an Apophis-sized (270 m) asteroid resulted in a reduction of impacting mass of 97% using a non-gravitational estimator. The effect of Earth's gravity was shown to be significant in accurately predicting the orbit of a fragmented asteroid on an impact trajectory. Further, the inclusion of mutual gravitational terms among the fragments was shown to reduce the expected effectiveness of fragmentation to 93%. The scenario examined showed the ability of the proposed model to handle multiple body interactions with significantly reduced computational time from previous models. Efficient algorithms for this problem were discussed as a motivation for researchers on a budget to consider parallelization of existing simulation codes. Use of open source libraries for parallelization resulted in quasi-linear speed up of computational resources and results applicable to research in the field of asteroid impact threat mitigation.

ACKNOWLEDGMENT

This research work was supported by a research grant from the Iowa Space Grant Consortium (ISGC) awarded to the Asteroid Deflection Research Center at Iowa State University. The authors would like to thank Dr. William Byrd (Director, ISGC) for his support of this research work. Part of this work by the third author was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.

REFERENCES

[1] J. Hennessy and D. Patterson, *Computer Organization & Design*. San Francisco: Morgan Kaufmann Publishers, 1998.

[2] M. P. I. Forum, "MPI-2: Extensions to the Message-Passing Interface," 2003.

[3] O. A. R. Board, "OpenMP Application Program Interface," Vol. 3, 2008.

[4] P. S. Pacheco and W. C. Ming, "MPI Users' Guide in FORTRAN," 1997.

[5] J. Hass et al., *University Calculus*. Boston, MA: Pearson, 2007.

[6] H. Schaub and J. L. Junkins, *Analytical Mechanics of Space Systems*. Reston, VA: AIAA Education Series, 2003.

[7] T. Gehrels (ed.), *Hazards Due to Comets and Asteroids*. Tucson, AZ: The University of Arizona Press, 1994.

[8] J. Sanchez et al., "On the Consequences of a Fragmentation Due to a NEO Mitigation Strategy," *59th International Astronautical Congress*, Glasgow, UK, 2008.

[9] B. Kaplinger and B. Wie, "Orbital Dispersion Simulation of Near-Earth Objects Deflection-Fragmentation by Nuclear Explosions," *60th International Astronautical Congress*, Daejeon, Korea, 2009.

[10] T. Ahrens and A. Harris, "Deflection and Fragmentation of Near-Earth Asteroids," *Nature*, Vol. 360, 1992, pp. 429–433.

[11] B. Wie and D. Dearborn, "Earth-Impact Modeling and Analysis of a Near-Earth Object Fragmented and Dispersed by Nuclear Subsurface Explosions," *20th AAS/AIAA Space Flight Mechanics Meeting*, San Diego, CA, 2010.

[12] S. Abe et al., "Mass and Local Topography Measurements of Itokawa," *Science*, Vol. 312, 2006, pp. 1344–1347.

[13] H. D. Curtis, *Orbital Mechanics for Engineering Students*. Oxford, UK: Elsevier, 2006.