5-2003

Thermal Evolution of Planetesimals and Protoplanets in the Terrestrial Planet Region: Code Optimization and Implementation on a Distributed Grid using NetSolve

Amitabha Ghosh
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Amitabha Ghosh entitled "Thermal Evolution of Planetesimals and Protoplanets in the Terrestrial Planet Region: Code Optimization and Implementation on a Distributed Grid using NetSolve." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Jack Dongarra, Major Professor

We have read this thesis and recommend its acceptance:

Allen J. Baker, Robert Ward

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate School:

I am submitting herewith a thesis written by Amitabha Ghosh entitled "Thermal Evolution of Planetesimals and Protoplanets in the Terrestrial Planet Region: Code Optimization and Implementation on a Distributed Grid using NetSolve". I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Jack Dongarra
Major Professor

We have read this thesis
and recommend its acceptance:

Allen J. Baker

Robert Ward

Accepted for the Council:

Anne Mayhew

Vice Provost and
Dean of Graduate Studies

(Original Signatures are on file with official student records.)

Thermal Evolution of Planetesimals and Protoplanets in the Terrestrial Planet Region: Code Optimization and Implementation on a Distributed Grid using NetSolve

A Thesis
Presented for the
Master of Science Degree
The University of Tennessee, Knoxville

Amitabha Ghosh
May 2003

To Daisy

“Because The World Is Round,
It Turns Me On….
Because The Wind Is High,
It Blows My Mind….
Love Is Old, Love Is New.
Love Is All, Love Is You.” (Abbey Road, The Beatles)

Acknowledgements

I would like to thank Dr. Jack Dongarra for motivating me in large part to pursue this degree, for his encouragement and for patiently listening to my ideas. I would like to acknowledge the help and friendship of some of the wonderful people I came to know in the Computer Science department. I owe a special word of thanks to Dr. Allen J. Baker, who steadfastly helped me with just about anything I needed. I would like to thank Dr. Robert Ward for his help and support.

Abstract

A code for asteroidal heat transfer and growth is optimized for performance. The Gauss elimination routine for the solver is replaced by a sparse matrix routine. Finite element matrix assembly operations are rewritten to reduce operations involving 3D arrays to 1D. Advantage is taken of the sparse matrix structure of finite element matrices in reducing 2D arrays to 1D. The number of vector touches are reduced to the extent possible, by carrying over statements from one iteration to the next. The number of do loops are reduced by merging several do loops into one. The optimization reduced the CPU time taken to run the code from 297 sec to 0.88 sec for a matrix size of 100, an improvement of 99.70%. More importantly, the algorithm was reduced from a $O(n^3)$ operation to a $O(n)$ operation. Thus, the percent time difference between the optimized and unoptimized versions is greater at larger matrix sizes. At matrix sizes of 100, the number of floating point operations were reduced from 2.39 E+09 to 2.99E+07, an improvement of 98.75% and the performance was increased by about 4 times, from 8.06 MFLOPS/s to 33.92 MFLOPS/s. Because of inefficiency in memory allocation, the maximum matrix size for the unoptimized code was limited to 200. This was increased to 5,000,000 for the optimized code. A version of the code was implemented on NetSolve and added to the list of problems on netsolve.cs.utk.edu. Two sample movies were generated using OpenGL to explain the scientific significance of the code. With the implementation of the optimized code, applications to address scientific problems can now be envisioned that were previously thought to be prohibitive in terms of computer time.

Table of Contents

List of Figures

I. Introduction

There is a crucial disconnect in the way a scientist or engineer and a computer scientist approaches scientific programming. To the scientist, what matters is the ultimate solution. A computer scientist on the other hand cares more about the intricacies of the code, including its structure, organization and performance. In the field of high performance computing, incredible strides have been made in the last two decades.

Yet, the tremendous increase in computing power, has hardly percolated to mainstream scientific computing. Scientists write code in a style resembling a mathematical derivation or an algorithm, without regard to performance issues. The primary motivation is the final solution: performance is hardly a consideration, until of course, the run time of the code is found to be unrealistically long. At this point, most scientists make mathematical approximations to simplify the complexity of the code, and use sample calculations or mathematical derivations to rationalize the simplification.

The resistance of scientists to high performance coding is due to the necessary overhead in understanding and implementing high performance codes. Standardized high level languages for high performance computing like HPF are available but the code needs to be converted from languages most scientific code is written in like Fortran and C. Message passing programming though available is tedious and hard to debug. Unless a direct scientific benefit is apparent, a scientist is unwilling to invest the time and effort to master performance issues. Another central issue seems to be whether high performance computing is at all necessary for most science applications. The answer to this is interesting: unless realistic in terms of computational time, a science application is not thought of or pursued. This results in a chicken-and-egg problem: because of the overhead of migration, a high performance version of the code is not developed. Since, high performance codes are not available, scientific applications that are computationally intensive are not perceived to be realistic and hence not pursued.

The use of optimized software libraries is a step forward in the evolution of high performance scientific code. Yet, there are certain disadvantages to software libraries. They do require a degree of programming for incorporation. Moreover, certain software resources maybe available only on specific platforms, or on specific machines on the network. The evolution of Distributed Computing allows a user to use disparate computational resources over machines distributed over the local area network or the internet. The evolution of grid middleware, like NetSolve, serves as a layer between the application and the available software/hardware interface. The middleware Application Programming Interface (API) thus, allows user to access aggregate resources over the grid, without a corresponding requirement to understand computing issues, like networking, load balancing, fault tolerance, etc. Through the use of grid middleware, application programmers can now hope to have access to aggregate computational resources without the need to actually understand underlying supercomputing issues. In NetSolve, a subroutine can hence be optimized and registered as a grid software resource. Once this is done, the subroutine can be accessed in any code (in C, Fortran, Matlab or Mathematica) from multiple client machines across the grid. The development of grid middleware, thus presents the minimum computational overhead for the scientist or engineer, to extract performance as well as access to heterogeneous machines, software libraries and large computational resources.

II. Overview Of Asteroid Thermal Models

Planetary Science: An Introduction

The Sun is believed to have formed by the gravitational self-collapse of a cold dense interstellar molecular cloud. Conservation of angular momentum precludes a molecular cloud from collapsing into a single object of stellar dimensions: thus, the solar nebula is envisioned as a single central condensation surrounded by a flattened, rotationally supported structure termed an accretion disk. Disks evolve by dissipation of gravitational energy, kinetic energy and angular momentum: material either lose angular momentum to fall into the protostar or gain angular momentum and move outward. Formation of planets takes place in the accretion disk either by gravitational collapse (gaseous giant planets) or due to accumulation as a result of two-body collisions (terrestrial planets). The terrestrial planets comprise about 0.5% of the mass of the planetary system, and the planetary system in turn corresponds to ~0.1% of the mass of the Solar System. Yet, our investigation of Solar System formation centers in large measure around the study of the evolution of the terrestrial planet region, primarily due to the availability of a wealth of observational and cosmochemical data about the terrestrial planets. The problem of uniquely inverting these observations to infer conditions in the early Solar System is beyond our abilities, primarily due to a lack of understanding of "forward problems" and the paucity of observations. Yet, the idealized pursuit of uniqueness, through interdisciplinary study of various processes, defines the ultimate goal of a Planetary Scientist. While "requests for the luxury of uniqueness are premature" (Wetherill, 1980), it is nonetheless important to strive to eliminate or resolve different conclusions arrived at through different modes of enquiry.

Motivation: An Identified Demand For Cross-Disciplinary Research In Planetary Science

There are two approaches in understanding the early history of the Solar System: the first involves theory based on mechanical and hydrodynamical models for the formation of the solar nebula (using observations of accretionary disks in other stellar systems), whereas the second focuses on materials shaped by processes in our own nebula, which range from the compositions and ages of components found in meteorites to observations about the Solar System as we see it today. To the extent that the different approaches address the same issue, they should yield consistent results. Yet, in reality, different methods often lead to conflicting conclusions that often stay unresolved because of the inadequate communication between workers across disciplines (Podosek and Cassen, 1994). Thus, there has been a growing recognition in Planetary Science of the need to communicate findings of a given discipline in a form that is accessible to other disciplinary scientists: which is the long term motivation behind the present work.

Thermal Models As A Tool For Cross-Disciplinary Research

A common cord that can be used to relate concepts as disparate as multibody dynamics of accretion and the geochronology of rocks is thermal modeling. Almost any process that relates to either asteroidal/planetary evolution or its cosmochemical record in extraterrestrial material (e.g. closure age and cooling rates of meteorites) can be traced to the heat budget of the relevant body. All figures and tables are attached in the Appendix at the end of the thesis. Fig. A1 shows a generalized sequence of events in the evolution of asteroids and terrestrial planets. Thus, accretion is followed by heating that causes either melting, metamorphism or aqueous alteration, followed by cooling, fragmentation and reassembly. In the terrestrial planet feeding zones, asteroid-size bodies act as planetary building blocks. This is a generalized diagram: it is possible that some stages are bypassed in certain cases (like the incorporation of planetesimals into planets can take place before cooling, fragmentation and reaccretion). It is of interest to note that each of the stages (in Fig. A1) effect the heat budget of the body. Thus, a specific thermal evolution scenario can give information about the various stages. On the other hand, a particular experimental observation (or theoretical constraint) can be tied to a specific thermal evolution scenario.

Thermal modeling has been used for the last three decades to study evolutionary histories of asteroids and meteorite parent bodies (Table A1). The heat transfer equation (for a detailed discussion, see Chapter III) is solved for an asteroidal body or a planetesimal. A Dirichlet boundary condition or a radiation boundary condition is used to simulate heat loss from the surface. Asteroid thermal models use parameters from accretion physics, models of asteroid fragmentation and nebular models in order to generate thermal evolution scenarios that can then be compared to meteorite and asteroid data. Thermal models use input parameters (like accretion rate and time, ambient temperature, degree of compaction, etc.) constrained from nebular models, accretion models, models of fragmentation and regolith formation, etc. The model strives to match the peak temperature constraint of the relevant type of body being modeled (For details, see McSween et al, 2003). The simulation provides as output, data like model cooling rates and model closure ages and volume proportion of petrologic types. These can be compared with cosmochemical data like meteorite cooling rates and closure ages. If the match is unsatisfactory, the model is recalculated with a different set of input parameters and the processes until a satisfactory match is obtained. In essence, thermal models map a set of input parameters to a corresponding set of output parameters by balancing the peak temperature. Thus, they can sequester parameter sets that are compatible with one another from those that are not. For example, in modeling the HED parent body, ^{26}Al chronology (relative to the formation of Calcium Aluminum inclusions in meteorites or CAIs) place volcanism at 2-3 Myrs (Srinivasan et al., 1999), whereas W-Hf systematics produces a time of core segregation between 6-15 Myrs (Lee and Halliday, 1999). Using thermal models, it is possible to assess that the older ^{26}Al ages requires accretion to initiate at CAI formation and a magma ocean to form on Vesta. The younger W-Hf age would require accretion to take place ~2.5 Myrs after CAI formation and melting on the HED parent body to be restricted to <25%. Thus, a thermal model can translate two disparate chronologic ages and translate them into two different whole asteroid evolutionary scenarios. In other words, thermal models can be used as a platform to compare and put into context the various interdisciplinary approaches of studying the early Solar System.

Probable Heat Sources: ^{26}Al And Electromagnetic Induction Heating

In 1955, Harold Urey recognized that "it is difficult to believe that heating by K, U and Th (or longlived radionuclides) is a feasible explanation for the high-temperature stage required to produce the meteorites." He proceeded to perform the first back of the envelope calculation where he suggested that ^{26}Al was a heat source for asteroidal heating (Urey, 1955), and perhaps inadvertently set into motion the subdiscipline of asteroid thermal modeling. For about the next 30 years, thermal models were used as plausibility calculations for heat sources where the main aim was to realize peak temperatures for chondrite metamorphism in an asteroidal body. Of the many heat sources suggested to cause global metamorphism in the early solar system, all but, heating due to decay of ^{26}Al and heating due to electromagnetic induction of asteroids, are considered implausible (See Wood and Pellas, 1991, for detailed discussion).

The theory of electromagnetic induction heating, first proposed by Sonnett et al. (1968) was based on the physics of T-Tauri outflow (Kuhi, 1964). However, recent studies of T-Tauri stars moderate the conditions required for electromagnetic induction heating. First, the solar wind has been found to be anisotropic with a higher wind density at high latitudes, avoiding the equatorial region where planetesimals form (Edward et al., 1987). Second, the mass loss from a T-Tauri star (and hence, rate of mass loss which governs the magnetic field of the plasma) has been revised from ~50% (Kuhi, 1964) to a few percent (DeCampli, 1981). The problem hinges on the choice of a reasonable parameter set where, as noted by Wood and Pellas (1991), most of the parameters are unconstrained. However, recent models (Herbert, 1989, Shimazu and Terasawa, 1995) do produce melting in asteroidal sized bodies. In the absence of any conclusive study to prove otherwise, we cannot rule out electromagnetic induction as a heat source in the early Solar System.

In contrast, he case for ^{26}Al has become increasing stronger over the last decade. ^{26}Al in the early solar system was widespread (MacPherson et al., 1995; Huss et al., 2002), and its decay product has been found in most classes of chondrites (Lee et al., 1976; Russell et al., 1996; Kita et al., 2000) and an achondrite (Srinivasan et al., 1999).

Reasons as to why evidence for ^{26}Al might be obscured in other achondrites have been given (LaTourrette and Wasserburg, 1997; Ghosh and McSween, 1998). This heat source appears capable of explaining the full range of temperature excursions of asteroids within the main belt (Grimm and McSween, 1993). Although nebular heterogeneity of ^{26}Al has been suggested (Ireland and Fegley, 2000), the consistency of ^{26}Al/^{27}Al ratios in calcium-aluminum-rich inclusions (CAIs) and in chondrules, regardless of chondrite class, implies broad nebular homogeneity and indicates that differences in initial ratios reflect formation time (Huss et al., 2001).

Laboratory studies of meteorites and spacecraft images of asteroids reveal the importance of impacts in the evolution of the asteroidal bodies (Keil et al., 1997). Impacts have been suggested as the heat source for global thermal metamorphism, igneous activity and aqueous alteration, as well as selective melting of asteriodal bodies (Wasson et al., 1987, Cameron et al., 1990, Rubin, 1995). Although hydrodynamic models show that some particles ejected in asteroidal collisions do indeed cause heating to metamorphic and melting temperature, most material does not undergo heating (Keil et al., 1997). Impacts are believed to have caused insufficient to cause global heating of entire asteroids: the temperature rise has been shown to be at best tens of degrees (e.g. Keil et al., 1997). This primarily stems from the fact that impact energy is proportional to gravitational potential energy that is negligible in bodies of asteroidal dimensions (Melosh, 1990). Selective melting has been proposed (Wasson, 1995 and references therein) for the proposed for the origin of silicate-bearing IAB, IIICD, and IIE iron meteorites. Impacts into megaregoliths of chondritic composition are postulated to cause Fe-FeS eutectic melting. The melt is thought of have separated from silicates that either remained unmelted or underwent selective melting and fractionation. Repeated impacts of the regolith are hypothesized to have formed separate metal melt pools. However, the melt pools are believed to have solidified almost instantaneously preventing melt movement or collection into pools chiefly due to the short duration of the pressure and temperature pulse in impacts (Keil et al., 1997 and references therein). Moreover, impacts do not produce partial melts: either instantaneous whole rock melting takes place or melting at the scale of mineral grains is observed (Stoffler, 1988).

Approaches To The Problem

There exist three methods for numerical solution of the heat transfer: the classical series solution (e.g. Miyamoto, 1981; Bennett and McSween, 1996; Ackridge et al., 1997), the finite difference method (e.g. Wood, 1979; Grimm and McSween, 1993) and the finite element method (e.g. Ghosh and McSween, 1998). The finite element method, which uses a basis function to minimize approximation error during numerical integration, has been found to be more accurate than either the finite difference method or the classical series solution (Baker and Pepper, 1991). Most models assume asteroidal accretion to be instantaneous. Ghosh and McSween (2000) presented a model for incremental accretion, and Ghosh et al. (2001) use results from an accretion model (Weidenschilling et al., 1997) in an incremental accretion thermal model. The thermal evolution of the asteroid can be roughly divided into accretion, heating (with or without melting and differentiation or aqueous alteration) and cooling. Workers for simplicity have simulated part of the process like (a) heating (with metamorphism) and cooling (e.g. Miyamoto, 1981), (b) heating (with melting and differentiation) and cooling (Ghosh and

McSween, 1998), (c) heating (with aqueous alteration) and cooling (e.g. Grimm and McSween, 1989) and (d) cooling (Haack et al., 1991).

Depending upon the choice of parameters, boundary and initial conditions, numerical method and stages of asteroidal evolution, a plethora of thermal models have been presented in the last twenty-five years. Table A1 summarizes the evolution of thermal models over the last fifty years.

III. Problem Description And Algorithm Development

Heat Transfer Equation

The fundamental heat transfer equation in spherical coordinates is given as follows:

$$\frac{dT}{dt} = \frac{1}{R^2}\frac{d}{dR}(R^2\kappa\frac{\partial T}{\partial R}) + \frac{Q_{radnl}}{\rho c_v}$$

where T is temperature, R, the radius of the asteroid, t=time, κ = thermal diffusivity, Q_{radnl} heat generated by radioactive decay, ρ = density, and c_v = specific heat at constant volume. To appreciate the subtleties of the model, it is useful to understand how each term in the equation might affect the solution. The term on the left side is the rate of change of temperature (T) with time (t) in a layer of infinitesimal thickness at any arbitrary depth in the asteroid. The first term on the right side of the equation gives the amount of heat gained in (or lost by) the infinitesimal layer from the surrounding layers by conduction and is known as the conduction or diffusion term. In words, it means that the amount of heat transmitted in this fashion is proportional to the rate at which the product of thermal gradient (dT/ dR) and thermal diffusivity (κ) changes. Thermal diffusivities of rocks are typically low (around 10^{-7} m^2 s^{-1}) and are functions of temperature. The thermal diffusivities, as a function of temperature, for various rock types are taken from Yomogida and Matsui (1983). The last term on the right side gives the amount of heat that is generated in the asteroid. Q_{radn}l varies with the live ^{26}Al and ^{60}Fe present and the absolute abundance of Al and Fe. It can be represented mathematically as:

$$Q_{radnl} = A_{0Al}Q_{0Al}e^{-\lambda_{Al}t} + A_{0Fe}Q_{0Fe}e^{-\lambda_{Fe}t}$$

where A_0 = initial abundance, Q_0 = initial heat production per unit volume and λ is the decay constant.

Finite Element Implementation

The simulation is based on the finite element method. The finite element method uses a basis function to minimize approximation error during numerical integration, and has been found to be more accurate than the finite difference method or the classical series solution (Baker and Pepper 1991). The mode of derivation of the Galerkin Weak Statement (and the subsequent matrix form of the equation) from the heat transfer equation is after Baker and Pepper (1991). The temperature is approximated by a trial function which does not coincide with the exact solution of the differential equation for a particular value of the spatial dimension, r.

Hence,

$T(r) = T^N(r) + e^N(r)$,

where $T(r)$ is the exact solution, $T^N(r)$ is the numerical approximation, and $e^N(r)$ is the error. One of the primary concerns of a simulation is to minimize the approximation error $e^N(r)$. An estimate of the error can be found by substituting it in the differential equation ($L(T) = 0$) to be solved.

Thus, $L(e) = L(T) - L(T^N)$

Since, $L(T) = 0$, $L(e^N) = -L(T^N)$

Thus, the trial function, when substituted in the differential equation does not equal zero, as required by the differential equation. This cannot be done because forcing

$L(T^N)$ to be equal to zero amounts to evaluating the exact solution. Instead, a rational approach is to make a measure of approximation error disappear in the overall integrated sense over the domain. This is done by setting the weighted residuals to go to zero over the entire domain represented by the following statement:

$$\int_{\Omega} w_i(x) L(T_N) = 0 \qquad \text{for } 1 \leq i \leq N$$

The weight function w_i is made identical to the trial function. This is called the Galerkin criteria that ensure the minimization of approximation error since it is orthogonal (in the mathematical sense means that the distance between the curves of the exact solution and the approximation is minimum) to the trial function. After implementing the weak statement, first-order Lagrange interpolation polynomials are chosen as the trial functions. Subsequently the trial function is written as a basis function for each generic finite element. The matrix statement is written and the boundary conditions are implemented at the two boundaries of the linear domain. Since a sphere is symmetric about a radius (and considering the fact that the asteroid does not have any directional heterogeneity in thermal properties), the formulation of the heat transfer equation in polar coordinates makes the problem one-dimensional. A time Taylor series is written on the matrix statement and the trapezoidal rule is implemented. The asteroid goes through a complex history of melting, differentiation, and cooling. The values of thermal diffusivity and specific heat are updated (as a function of temperature) for each time step.

$T^h + e^h = T_{exact} = T^{h/2} + e^{h/2}$, where T_h and $T_{h/2}$ are the solutions for a particular mesh size and its double mesh refinement, respectively, e^h and $e^{h/2}$ the errors for a particular mesh size and its double mesh refinement, respectively, and T_{exact} the exact solution. Approximation error has the following functional form (Baker and Pepper 1991):

$$e^h = C_k l_e 2k$$

Therefore, the relation between T^h and $T^{h/2}$ can be written as follows:

$$T^h - T^{h/2} = (2^{2k} - 1)e^{h/2}$$

Assuming $\Delta T_h = T_h - T_{h/2}$ can be written as follows, the error in the finer grid solution can be written as

$$e^{h/2} = \Delta T^{h/2}(2^{2k} - 1)$$

The slope of a log–log plot of numerical error and length of a finite element (l_e) would be expressed as:

slope = $[\log(e^{h/M}) - \log(e^{h/2M})]/[(\log(l_e) - \log(l_e/2))]$

For ideal convergence, the above expression should theoretically be equal to 2 for a linear basis approximation. Progressive mesh refinements were performed and the value of slope was found to equal to 2.05 (see Table A2).

Boundary Conditions

The heat flux at the center of the asteroid is assumed to be zero. The heat loss from the surface of the asteroid is governed by the radiation boundary condition and given by

$$\frac{dT}{dR}\Big|_{R=R^{"}} = \frac{e\sigma}{k}(T_{surf}^4 - T_{nebula}^4)$$

where R' is the radius at the surface, T_{surf} temperature at the surface of the asteroid, T_{nebula} temperature of the surrounding nebula, e emissivity, and σ Stefan Boltzmann constant. T_{nebula} is assumed to be the same as the initial temperature throughout the time domain. The temperature of the nebula probably decreased with time, but at this time the rate of decrease is not unknown. Also, theoretical simulations, the basis of such calculations, are not anchored to meteorite evidence and cannot be standardized to a timescale relative to CAI formation, and as a result cannot be compared to the present asteroidal model (which is anchored to CAI formation).

Grimm and McSween (1989), Miyamoto et al. (1981), etc., use a Dirichlet boundary condition according to which the temperature at the surface of the asteroid is set to be the same as the temperature of the surroundings. A Dirichlet boundary condition will result in lower peak temperatures and higher cooling rates. The deviation is proportional to the difference in surface temperature as calculated by the radiation boundary condition and the temperature of the nebula.

Algorithm Development

The diffusion equation for heat transfer is solved using the finite element method for a symmetric spherical body. Linear 1-dimensional finite element basis matrices are used. A radiation boundary condition is used at the surface of the spherical body. Briefly, algorithm development can be summarized as follows: the Galerkin Weak Statement is written from the original equation as outlined in Baker and Pepper (1991). The weak statement is then transformed into a matrix statement of the form:

$$W_s = [M]\frac{d\{Q(t)\}}{dt} - [K]\{Q\} - \{b\} = 0$$

where, [M] and [K] = Square Matrices, and{Q}and {b}= Column Matrices.
The Taylor Series is then written on the matrix statement to evaluate the temperature matrix {Q} at time t+1 given all matrices, given {Q} from time t, i.e. the previous timestep.

The sample problem has two temporal domains: for the first time domain (6.6 Million years [Myrs]), a moving boundary condition is used. At each step, the radius of the spherical body is increased and the finite element domain remapped. The radius of the spherical body increases from an initial radius of 10 km to a final radius of 90 km. In the second time domain (3.4 Myrs for present purposes), the radius of the body does not change. There are 3 time domains following time domain-2 if the asteroid if asteroidal temperatures are high enough to cause melting to deal with metal segregation and volcanism on the asteroid. In the present case, a scenario is assumed where the asteroid does not melt.

The heat source for the system is the decay of ^{26}Al, with a half life of ~0.72 Myrs. As the timeframe of the simulation is ~10 Myrs or ~14 half lives, the heat generated decreases by a factor of 2^{-14} during course of the simulation. The specific heat capacity is the weighted sum of the specific heats of constituent minerals that make up the rock. The specific heat of each mineral is a function of temperature. Thermal diffusivity is a function of temperature.

The original code has 45 finite elements in the first temporal domain, and 85 finite elements in the second temporal domain. The first and second temporal domains have 4000 and 300 timesteps, respectively. The code reads in and non-dimensionalizes the values of the physical parameters. The code initializes and assembles the finite element basis matrices for each domain. The matrix equation is then solved by Gauss elimination.

The algorithm can be summarized as follows:

i) Basis matrix assembly for each finite element

ii) Assemble [M], [K] and {b}from basis matrices

iii) Using Taylor Series, construct input matrix for Gauss elimination from matrices in (ii)

iv) Solve input matrix and obtain temperature at time t as output

v) Use temperature at time t to reassemble matrices [K] and {b} (and [M} for time domain-1), i.e. go to (iii)

In this thesis, various measurements are made by varying the number of finite element domains: this changes the size of the finite element matrices. Thus, matrix size as referred to in this paper, refers to the number of finite element nodes, i.e. a matrix size of 50 means that the number of finite element nodes is 50.

Measuring Time, Flops And MFLOPS/s

Performance data for the code is obtained using PAPI for torc9.cs.utk.edu. PAPI stands for Performance Application Programming Interface, and is developed at the Innovative Computing Laboratory at the University of Tennessee (PAPI website, 2003; PAPI User Guide, 2003; London et al., 2001a; London et al., 2001b; Dongarra et al., 2001; Browne et al., 2000a; Browne et al., 2000b; Browne et al., 2000c). The project implements an API (Application Programming Interface) to access hardware performance counters of various microprocessors. This is developed in part to enable application developers to identify code inefficiencies: so it is well suited to identify code inefficiencies in the present code for asteroidal heat transfer. Hardware counters exist on almost all platforms: the advantage of PAPI is that it is portable across multiple platforms. For more information, see icl.cs.utk.edu/papi and the PAPI User Guide.

The function PAPIF_flops in Fortran (corresponding to PAPI_flops in C) is used to measure the total process time in seconds, the number of floating point operations and MFLOPS/s. Unless specified otherwise, the measurements were made on torc9.cs.utk.edu (Operating system: Linux, Memory= 256 MB , Processor = 600 MHtz, Pentium III with 512 KB L2 cache).

The function PAPIF_flops is unavailable in cetus4a.cs.utk.edu since it has an UltraSparc processor. Ultrasparcs do not support PAPI_FLOPS. PAPI supports any event the processor supports. Thus, since the PAPIF_flops call uses the event PAPI_FLOPS, this does not run on Cetus4a. PAPIF_flops works on SGI, Linux (Except AMD Athlons), Unios (Cray T3E), Windows (Except AMD Athlons), Itanium 1 & 2 and AIX boxes.

Since the function PAPIF_flops is unavailable in Unix boxes, time is measured using the *time* function in fortran (elapsed (1) and elapsed (2) give the system and user CPU time) for cetus4a.cs.utk.edu (Operating System=Unix, Memory= 512 MB, Processor= 500 MHtz, UltraSparcIIe with 256 KB L2 cache). The System CPU time + User CPU time required for execution is measured for different problem sizes.

Unless specified otherwise, time data is reported for cetus4a.cs.utk.edu and flops and MFLOPS/s data are reported for torc9.cs.utk.edu.

IV. Replacing The (Gauss Elimination) Routine With A Sparse Tridiagonal Solver

Exploiting Matrix Structure For Performance

The efficiency of any matrix algorithm depends on multiple factors. One of the most intuitive factors is identifying the amount of redundant arithmetic and storage for a given matrix algorithm. It is important to exploit matrix structure, particularly in case of sparse matrices, to optimize performance. Thus, matrix structure can be exploited for efficient storage and to reduce the number of redundant arithmetic operations. Specifically, the properties of bandedness and symmetry can be exploited to increase algorithm efficiency.

Solver Used In Original Code

In the heat transfer code, a system of linear equations need to be solved at every timestep. The solver routine used for this purpose is Gauss elimination. Since, the solver routine is invoked at every timestep, it is responsible for a considerable proportion of the floating point operations. Thus, optimization of the solver routine can yield a considerable boost to performance.

The broad motivation behind Gauss elimination is to convert a given system of equations: Ax = b, to an equivalent triangular system. During the transformation process (to generate a tridiagonal system), partial pivoting is used to prevent error magnification. This upper triangular system is then solved by back substitution. Like matrix multiplication, it is a triple loop process. Thus, as in matrix multiplication, a block LU algorithm can be developed that will enhance performance. In terms of performance, Gauss elimination is expensive, i.e. it is of the order $O(n^3)$, the number of flops being proportional to $2n^3/3$.

Using A Tridiagonal Solver

The matrix that was being solved using Gauss elimination was found to be tridiagonal and symmetric. This is because linear basis finite element matrices are 2 X 2 square matrices. The finite element assembly of these matrices throughout the domain produces a tridiagonal matrix. Since the coefficients of matrix element (2,1) and (1,2) for each of the finite element basis matrices are equal, the coefficients of the lower and upper diagonals are the same: thus, resulting in a symmetric matrix. When the matrix A of the system Ax = b is symmetric as well as positive definite, pivoting is not necessary. This enables solutions that are elegant, as well as compact. Using the symmetric, positive definite and tridiagonal nature of matrix A, a sparse tridiagonal matrix solver was implemented from Golub and Van Loan (1996).

Consider the system of [A]{x}= {f}, where [A] is n X n matrix and {x} and {f} are column matrices. The Gauss elimination code uses [A] and {f} as inputs. For the sparse tridiagonal solver, the diagonal of the matrix [A] was written as an array a (n) and the lower diagonal was written similarly as an array b (n-1). Since, matrix [A] is symmetric, the lower diagonal equals the upper diagonal. The underlying motivation of the algorithm is to reduce the tridiagonal system to an upper triangular system, and then using back substitution to compute the vector {x}. The new solver routine, thus takes {a}, {b}, {f} as inputs and solves for {x}. To optimize memory used, the vector {f} used, as input is overwritten with {x}, the solution and returned as the output. Gauss elimination was $O(n^3)$, but the new solver is $O(n)$, the number of flops being proportional to 8n.

Result Of Replacing The Solver

The implementation of the solver causes significant improvement in code run time. A comparison of the decrease in the run time of the Gauss elimination solver and the tridiagonal solver for various matrix sizes is shown in Fig. A2, A3 and A4. The overall results are summarized in Fig. A5. Each of the figures A2 – A5 plots CPU time on the Y-axis on a logarithmic scale. The X-axis plots the last 250 of the 4000 timesteps of time domain-1 followed by the first 250 timesteps of time domain-2. Thus, the left halves of these plots represent time domain-1 where matrix size is kept constant 45. The right half of the plot represents the first 250 timesteps of time domain-2: here the matrix size is varied from 99 to 149 to 199. The run time of the solver decreases from 8.81E-05 seconds to 4.88E-03 sec., an improvement of 98.2% for a matrix size of 199. More importantly, the solver reduces from a $O(n^3)$ operation to a $O(n)$ operation: thus, as n (where, size of A is (n,n)) increases from 46 to 199, the CPU time for Gauss elimination increases almost by two orders of magnitude. Since Gauss elimination is $O(n^3)$, increasing n from 46 to 199 (>4 times) should cause the time to increase by $199/46=4.32^3= 81$ which is broadly compatible with the results obtained in Fig. A5. CPU time for the new solver on the other hand is about a order of magnitude lower than that of Gauss elimination for n=46. More importantly, the increase in CPU time is not noticeable (particularly on a logarithmic scale for time in Fig. A2 – A5) for the sparse matrix solver since the solver is $O(n)$, whereas Gauss elimination is $O(n^3)$.

A marked change in performance of the solver is achieved by substituting the tridiagonal solver: this is reflected in the performance for the overall code. Thus, for a matrix size of 199, the number of floating point operations for the overall code decreases by 30.8% (Fig. A6). This difference is comparatively lower for smaller matrix sizes as shown in Fig. A6. As explained in the previous paragraph, this is because the old solver routine was $O(n^3)$ and the new solver is $O(n)$.

Fig. A7 summarizes the MFLOPS/s versus problem size for the (entire) code with the Gauss elimination solver (Series–1) and the Sparse Matrix solver (Series-2). A slight degradation in performance of 2 MFLOPS/s is observed. This can be partly attributed to the sparse tridiagonal solver used in place of the Gauss elimination routine. The algorithm is such that it cannot be vectorized. A sample loop from the algorithm runs like this:

```
do k = 2:n
     b(k) = b(k) - e(k-1)*b(k-1)
end do
```

It is clear that unless b(k-1) is known, b(k) cannot be evaluated. Since all do loops in the algorithm are of this nature, this degrades performance. This results in a slightly lower MFLOPS/s measurement for the optimized code with the sparse tridiagonal solver. Because of the dependence, the calculation in the above loop has to be performed serially and cannot be vectorized.

Memory Utilization

Figures A2 – A6 show performance results upto matrix sizes of 200. This is because jobs with a matrix size >200 were killed by the operating system for want of memory. The machine the runs were attempted on has 256 Mbytes and 512 Mbytes of memory, for torc and cetus, respectively. The memory needed for a square matrix of dimension 500 X 500. For 25,000 real numbers, the total memory required should be 200kbyte. So, accommodating a matrix of this size in memory should be trivial. This points to inefficiency in memory allocation. The code contains several 2D matrices and 3D matrices that use up a lot of memory. These matrices have been defined for algorithmic clarity. Functionally, some of these matrices are redundant and some others can be rewritten efficiently based on the matrix structure. In the later chapters, the issue of memory allocation is addressed and the code is optimized for performance.

V. Reducing All Finite Element Matrices To The Corresponding Sparse Matrix Form

Limitations Of Previous Iteration

During the previous iteration, a Gauss elimination routine was replaced with an efficient tridiagonal solver. When compared in isolation, the tridiagonal solver has a considerably lower run time and fewer floating point operations compared to Gauss Elimination. However, the reduction in run time for the entire code was as little as 5%. Fig. A8 summarizes the results. For a matrix size of 150, the run time (of the entire code) decreases from 2381 to 2289 seconds, a decrease of 92 seconds or a decrease of 3.86%. As is clearly apparent from Fig. A8, the solver routine causes a reduction in run time for the entire code, but the improvement is insignificant (i.e. by 3.86% at matrix size of 150). The decrease in the number of floating point operations is slightly greater (Fig. A6). One of the reasons for the insignificant decrease in run time is that the optimized version with the sparse solver reduces the number of flops, but at the same time it decreases the MFLOPS/s: thus, the decrease in flops is somewhat offset by the corresponding decrease in the performance as measured in MFLOPS/s.

Significant performance optimization of a serial code can be achieved by exploitation of the matrix structure: specifically, the properties of bandedness and symmetry. In the previous chapter, matrix operations involving the solver were optimized. In the present chapter, finite element matrix operations will be optimized. Specifically, finite element basis matrix generation and matrix assembly operations will be optimized.

Bandedness And Symmetry Of Finite Element Matrices

Three finite element square matrices are formed at the end of the assembly process. These are matrices [K], [M] and {b}: for further details on these matrices, see Chapter 3. [K] and [M] are generated by the assembly of linear basis finite element matrices that are 2 X 2 matrices. The finite element assembly of these matrices throughout the domain produces a tridiagonal matrix. A tridiagonal matrix of N X N dimension can be effectively stored as three arrays: an array each for the diagonal, the upper diagonal and lower diagonal. Since the coefficients of matrix element (2,1) and (1,2) for each of the finite element basis matrices are equal, the coefficients of the lower and upper diagonals should be the same: thus, resulting in a symmetric matrix. Thus, for the matrices [K] and [M], the lower diagonal is equal to the upper diagonal. Thus, a tridiagonal, symmetric matrix of dimension N X N can be stored as two arrays of size N and N-1 respectively, instead of a 2D array of N^2 numbers.

Improvement In Run Time

After implementing this phase of optimization, significant improvements in run time was observed. Thus, for a matrix size of 85, the run time decreased from 499 seconds to 1.66 seconds: a decrease of 99.7%. The run times as a function of matrix size for this phase of optimization is shown in Fig. A9. A corresponding comparison of the run time versus matrix size between the present and past iterations is summarized in Fig. 10.

Reduction In Floating Point Operations: Transition Of The Code From O(n^3) To O(n)

Fig. A11 summarizes the reduction in floating point operations achieved during this iteration. Note that the number of floating point operations at matrix size 100, is an order of magnitude lower than the previous iteration. Perhaps, a more important result is seen in the slope of each of the curves in Fig. A11. Thus, the rate of increase of flops with matrix size is far gentler after the present iteration. This is because for the previous version, since it is a $O(n^3)$ operation, the slope is steep. For the present version, the slope is gradual, and although not discernable in a logarithmic plot, increases proportionately with increase in matrix size: therefore, the present version is $O(n)$. This relationship will be more apparent in later figures (e.g. Fig. A12: with a linear scale on the X-axis that plots run time with matrix size).

Memory Optimization

Some memory optimization did take place at this stage. Thus, the maximum problem size increases from 200 to 2000. Still, this is not an adequate improvement given that the total available memory is 256 and 512 Mbytes on torc and cetus, respectively. Thus, memory optimization will again be addressed in the following chapter.

VI. Memory Optimization: Optimizing The Matrix Assembly Process And Replacing All 2D And 3D Matrix Arrays To 1D Form

Limitations Of Previous Iteration

In the last version, finite element matrices that were identified to be sparse were expressed as a linear arrays and a tridiagonal solver was implemented in place of a Gauss elimination routine. It was possible to run problems with matrix sizes upto 2000. Sizes >2000 could not be run on the Cetus machines because the memory requirements of the code exceeded available memory on the machine. This iteration deals with implementation of efficient memory management with the aim of attaining greater performance together with the ability of running larger problem sizes.

The Matrix Assembly Process

In the present iteration, the matrix assembly process to generate the finite element matrices [K], [M] and {b} are rewritten to ensure that there are no two-dimensional or three-dimensional arrays. The matrix assembly process is both computationally and memory intensive. The assembly process for [K] and [M] entails the use of 3D matrices and three nested do loops that imply operations proportional to n^3. For the column matrix {b}, the matrix operation process requires 2D matrices and two nested do loops: so the major drain on resources is the assembly of the square matrices. An optimization that collapses the do loops, and a formulation that avoids the use of 3D matrices should serve to improve memory utilization and performance. After this iteration all matrices are expressed as one-dimensional arrays is sparse matrix form, where the diagonal and lower diagonal are stored as arrays of size n and n-1, respectively.

Changes In Run Time, Memory Utilization, Flops And MFLOPS/s.

The run time of the code in this iteration for a problem size of 1000, was reduced from ~ 12 seconds to ~4 seconds, a reduction of 66%. The run time in seconds as a function of problem size for the present iteration is shown in Fig. A12. Fig. A13 shows the comparison in run time for the present iteration compared to the previous iteration. In contrast to Fig. A12, the X-axis for Fig. A13 is not linear but logarithmic.

Significantly, the maximum problem size that could be run on Cetus increased from 2000 to 5,000,000, i.e. 2500 times: thus, as shown in Fig. A12 and A13, runs for large problem sizes could be undertaken. The number of floating point operations decreased significantly as shown in Fig. A14. Note that the Y-axis of Fig. A14 is logarithmic. Thus, at matrix size 1000, the floating point operations for the previous version is 9.49E+08. For the present version, the number of floating point operations is 1.46E+08: thus, the number of floating point operations were reduced by 85%. Moreover, the problem is *O*(n) and thus the increase in matrix size causes almost a linear increase in run time as shown in Fig. A12.

A marked increase in MFLOPS/s is noticed. For the present version, with a problem size of 100, the speed is 32.94 MFLOPS/s (gradually decreasing to 26 MFLOPS/s for matrix size of 100000), compared to a value of 8.06 MFLOPS/s. The collapsing of the multiple nested do loops to a single do loop was probably responsible for a degree of vectorization and contributed in part to the increase in the MFLOPS/s. measured for the optimized code.

VII. Integration Of Do Loops To Enable Cache Reuse And Reduction Of Vector Touches

Vector Touches And Code Performance

In a matrix algorithm, an important consideration in terms of optimization is the amount of data that are moved around in the code. Data is moved around in chunks, and the time required to read or write a vector (referred to as a *vector touch,* defined as a vector load or store) to memory is significant. Rewriting the code such that the number of vector touches required is reduced causes significant savings in time to access vector data and can cause significant improvement in overall code execution. Instead of updating (and thereby accessing) a vector frequently, updates of a vector element can be written to a temporary scalar variable. The scalar can then be used to update the vector element at the end of the iteration. This lowers the number of vector touches and optimizes cache utilization since the vector has to be loaded (to be written or read) once. Thus, instead of loading a vector into the cache from memory multiple times, it is provident to minimize the number of loads from memory and maximize the number of times the data already loaded into cache is reused.

Cache Reuse And Code Performance

In addition, the structure of the code was changed to maximize use of data loaded into the cache. Cache reuse and utilization is one of the basics of generating high performance code. The objective is to use data loaded in the cache multiple times, instead of reloading the data from memory into the cache (since the time overhead of accessing the cache is much smaller compared to the time overhead of accessing memory).

Optimization To Maximize Cache Reuse And Minimize Vector Touches

In the asteroid heat transfer code, several do loops (11 to be precise) were merged into one do loop. The do loops cover different ranges: thus, the code had to be suitable modified to take care of this issue in the unified do loop. For example, some do loops start at 1 and continue until n+1, whereas others continue until n. Thus, the iteration n+1 was hard coded outside the loop. This could as well have been incorporated inside the loop with conditional statements (i.e. if (i= n+1) do …..), but this causes the code to evaluate each of these conditional statements, where in most cases the operation will not be performed, causing a greater overhead. Loop unrolling was used in some do loops. Statements because of scientific clarity were written in several stages (like A= B * C, A = A * D, A= A / E, were rewritten as A= B * C * D / E). Blocks were tested in do loops, to ensure maximum reuse of data loaded in the cache. In contrast to matrix multiplication, where blocking is effective in optimization, since in this case, the arrays are one-dimensional, blocking seemed to have little effect on performance.

Improvement In Run Time, Flops And MFLOPS/s.

The percent improvement in run time with matrix size is caused by better utilization of data loaded in the cache. Thus, at smaller matrix sizes, there would be proportionately smaller number of cache misses: thus, the present optimization that reuses data loaded in the cache causes a lower proportionate improvement for smaller matrix sizes compared to larger matrix sizes where the optimization causes a larger proportionate improvement in cache utilization. Thus, the improvement in time is ~8% at matrix sizes of 1000, and 18% at matrix sizes of 1,000,0000 (Fig. A15). The number of floating point operations decrease between 4 – 14% as shown in Fig. A16. Fig. A17 shows that a marginal improvement in the speed measured in MFLOPS/s between 3 (at matrix size=100) – 15% (at matrix size = 100000).

VIII. Summarizing The Optimization

Comparison Of The Original Code With The Optimized Version

To summarize, the Gauss elimination routine for the solver is replaced by a sparse matrix routine. Finite element matrix assembly operations are rewritten to reduce operations involving 3D arrays to 1D. Advantage is taken of the sparse matrix structure of finite element matrices in reducing 2D arrays to 1D. The number of vector touches is reduced to the extent possible, by carrying over statements from one iteration to the next. The number of do loops are reduced by merging several do loops into one. The optimization reduced the CPU time taken to run the code, for a matrix size of 100, by 99.70% as shown in Fig. A18. More importantly, the algorithm was reduced from a $O(n^3)$ operation to a $O(n)$ operation: as the scale for the X-axis is logarithmic in Fig. A18, the difference in slope between the original and optimized versions is not apparent. This is shown better in Fig. A13, which is a plot of run time as a function of matrix size. Thus, the percent time difference between the optimized and unoptimized version is even greater at larger matrix sizes. The optimization in terms of reducing floating point operations is shown in Fig. A19. Thus, at matrix sizes of 100, the number of floating point operations were reduced from 2.39 E+09 to 2.99E+07, an improvement of 98.75%. The performance of the code as measured in MFLOPS/s. is shown in Fig. A20. The performance is seen to increase by about 4 times, from 8.06 MFLOPS/s to 33.92 MFLOPS/s. Because of inefficiency in memory allocation, the maximum matrix size of the unoptimized code was limited to 200. For the optimized code, the problem size was increased to 5,000,000.

Scientific Transparency Of Optimized Code

A major issue to the application programmer is whether the optimized code remains structured enough for ease of understanding. The optimized code with the implementation of sparse matrices for the solver and the finite element matrix operations is transparent and easy of understand. However, reduction of several do loops into one to reduce the number of vector touches, makes the code harder to read or understand. This is because all finite element matrix operations: from basis matrix generation, to finite element assembly and Taylor series are all compressed in a giant do loop. Also, the several do loops that are condensed into one, straddle slightly different iteration ranges, adding to the lack of transparency. In this case, some hard coding was required to be able to merge the loops. This integration of do loops, has a positive effect on code performance, but does compromise code clarity and readability.

IX. NetSolve Implementation

Grid Computing And Netsolve

One of the objectives of grid computing is to create a virtual computer out of a large collection of heterogeneous systems sharing various combinations of resources. This entails standardization of sharing of heterogeneous resources. Grids help users manage problems of resource availability, performance, and data storage. Grids are an emergent paradigm in computer science and specifically address the transparent use of non-local resources by researchers. In addition, Grids promise to deliver to end users far more power than is available in any single supercomputing installation There are several emerging Grid platforms (e.g. Globus, Legion, Nimrod). In the present study, NetSolve is used for implementation of the code on the distributed grid. For details on the Netsolve platform, please see: http:///www.cs.utk.edu/netsolve. NetSolve has well-defined interfaces to high-performance linear algebra libraries and has been proven as a production platform with problems like IPARS and MCELL (Casanova and Dongarra, 1997; Arnold, Casanova and Dongarra, 1998; Arnold et al., 2002; NetSolve website, 2003). In addition, NetSolve can leverage other Grid platforms, like Globus, to provide extensive computational resources.

Implementation In Netsolve

Using the problem description (pdf) file wizard on the NetSolve website (icl.cs.utk.edu/netsolve), the pdf file of the problem was created. The subroutine was copied to the directory NETSOLVE_ROOT/src/SampleNumericalSoftware and was compiled with the makefile in the directory to generate a library (.a) file. The file was then uploaded (using the script of the NetSolve webpage) on the NetSolve agent (netsolve.cs.utk.edu) with hydra4d.cs.utk.edu as a server. (It was necessary to remove the lines in the pdf file starting with –L for it to work with Fortran). The fortran subroutine is of the form:

```
Subroutine stratf77 (nodes, max)
integer nodes
double precision max
```

It takes as argument the number of nodes, and returns the maximum temperature attained in any finite element node over the timeframe of the simulation. Time was measured by the time function in Unix. The results as a function of matrix size and time are summarized in the figure below. Note that the X-axis is logarithmic.

Results

After implementation on the distributed grid, the run time of the code is of the same order as of a serial run of the optimized code (Compare Fig. A21 with A18). As expected, a single run of a serial code on a distributed grid shows a slight decrease in run time (~10% for a problem size of 1,000,000) because of the overhead of communication in the grid development.

A plethora of science applications of this routine can be visualized where the computational resources of the grid could be harnessed. For example, calculating the thermal history of the asteroid belt would entail tracking the heat balance for 10^{12} bodies, and as many runs of the code. One of ways, such large-scale applications of asteroid (and planetary) thermal modeling can be feasible, is the availability of this routine on a computational grid environment. More details about possible science applications of the asteroid thermal evolution code is discussed in Chapter XI.

X. Visualization Using OpenGL

A short movie was generated in OpenGL to illustrate the broad scientific results of the study. Temperature was plotted against distance from the center of the asteroid and many such plots over time were merged to make an mpeg movie file. Movie 1 (www.cs.utk.edu/~aghosh/ghosh_movie1.mpeg) shows the evolution of asteroid temperature with growth (counterclockwise from top left) for Cases-1, 4, 6, and 2. The time for asteroidal growth decreases from Case-1 through Case-6. It is interesting to note that Cases-1 and 2 attain the highest temperature during the period of growth: previously, it was thought that asteroids attain their highest temperature after asteroidal growth terminates. The volume fraction of the coldest material is highest in Case 1 and lowest in Case 6.

Movie 2 (www.cs.utk.edu/~aghosh/ghosh_movie2.mpg) shows the comparative thermal evolution with and without a regolith. (Regolith is a thin layer of ultrafine soil that is formed by meteorite impacts and subsequently churning of the soil on atmosphereless bodies like the Moon, and asteroids.) The movie shows the growth and temperature history for (counterclockwise from top left) Case 1, Case 6, Case 6 with regolith and Case 1 with regolith. Regolith is added after the body stops growing. Note that there is a big difference in the thermal history on adding a regolith to Case 6 since the regolith is added to the asteroid when the heat source is potent. In Case 1 where regolith is added after the heat source is virtually dead, there is no difference in thermal evolution.

XI. Scientific Applications

Numerical modeling to address scientific questions is necessarily limited by the computational constraints. Thus, important problems that could be better understood but are prohibitive, in terms of computing time and memory requirements, are not pursued because such an attempt is thought to be unrealistic. The generation of an optimized code does not just help in handling better the problem for which it is developed: it also can bring along new applications that could not be thought of previously.

The present code is used to study the thermal history of 6 Hebe, an asteroid. The optimized code was conceived in order to pave the way for an accurate thermal evolution model of Mars, a body with 30 times the radius of Hebe and with greater evolutional complexity including volcanism and core separation. In addition to achieving this end, the code can also be applied to studying the thermal history of the asteroid belt, which initiates with 10^{12} bodies of radius 1 km and grows by collision due to mutual attractions. In a multivariate problem, the output is dependent on the values of multiple input parameters, the values of which may be bracketed by an error bar. Thus, it is often of interest to bracket the level of uncertainty of the output, given the uncertainty of input parameters. Such problems can be computed elegantly in the Grid environment using NetSolve.

List of References

References

Akridge G., Benoit P. H., and Sears D. W. G. (1998) Regolith and megaregolith formation of H-chondrites: Thermal constraints on the parent body. *Icarus*, 132, 185-195.

Arnold, D. C., Casanova H., Dongarra, J. (1998) Innovations of the NetSolve Grid System, *Concurrency- Practice and Experience*

Arnold D., Agrawal S., Blackford S., Dongarra J., Miller M., Seymour K., Vadhiyar S., Sagi K., Shi Z (2002) User's Guide to NetSolve v1.4.1. *ICL Technical Report No. ICL-UT-02-05 June, 2002.*

Baker, A. J. and Pepper D. W. (1991) Finite Element 1-2-3, McGraw Hill, New York.

Bennett M. E., and McSween H. Y. Jr. (1996) Revised model calculations for the thermal histories of ordinary chondrite parent bodies. *Meteorit. Planet. Sci.*, 31, 783-792.

Browne S., Dongarra J., Garner N., London K., and Mucci P. (2000a), A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters, *Proc. SC 2000, November 2000.*

Browne S., Dongarra J., Garner N., London K., and Mucci P. (2000b) A Portable Programming Interface for Performance Evaluation on Modern Processors. *University of Tennessee Technical Report, Knoxville, Tennessee, July 2000.*

Browne S., Dongarra J., Garner N., Ho G., and Mucci P. (2000c) A Portable Programming Interface for Performance Evaluation on Modern Processors. *The International Journal of High Performance Computing Applications* **14:3**, 189 - 204.

Cohen B. A. and Coker R. F. (2000) Modeling of liquid water on CM meteorite parent bodies and implications for amino acid racemization. *Icarus*, 145, 369-381.

DeCampli W. M. (1981) T Tauri winds. *Astrophys. J.*, 244, 124-146.

Dongarra, J., London, K., Moore, S., Mucci, P., Terpstra, D. (2001) Using PAPI for Hardware Performance Monitoring on Linux Systems. *Conference on Linux Clusters: The HPC Revolution, Urbana, Illinois, June 25-27, 2001.*

Fujii N., Miyamoto M., and Ito K. (1979) The role of external heating and thermal metamorphism of chondritic parent body. *Planet. Sci.,* 1, 84.

Ghosh A. and McSween H. Y. Jr. (1998) A thermal model for the differentiation of asteroid 4 Vesta, based on radiogenic heating. *Icarus*, 134, 187-206.

Ghosh A. and McSween H. Y. Jr. (2000) The effect of incremental accretion on the thermal modeling of Asteroid 6 Hebe. Meteoritics and Planetary Science, A59.

Ghosh A. Weidenschilling S. J., and McSween H. Y. Jr. (2001) Thermal consequences of the multizone accretion code on the structure of the asteroid belt. In *Lunar and Planetary Science XXXII*, CD #1760. Lunar and Planetary Institute, Houston.

Ghosh A. and McSween H. Y. Jr. (2003) Importance of the accretion process in asteroidal thermal evolution: Heat transfer in asteroid 6 Hebe. *Meteoritics and Planetary Science*, in Press.

Golub, G. H. and Van Loan, C.(1998) Matrix Computations.

Grimm R. E. (1985) Penecontemporaneous metamorphism, fragmentation, and reassembly of ordinary chondrite parent bodies. *J. Geophys. Res*., 90, 2022-2028.

Grimm R. E. and McSween H. Y. Jr. (1989) Water and the thermal evolution of carbonaceous chondrite parent bodies. *Icarus*, 82, 244-280.

Grimm R. E. and McSween H. Y. Jr. (1993) Heliocentric zoning of the asteroid belt by alumimun-26 heating. *Science*, 259, 653-655.

Haack H., Rasmussen K. L., and Warren P. H. (1990) Effects of regolith/megaregolith insulation on the cooling histories of differentiated asteroids. *J. Geophys. Res.*, 95, 5111-5124.

Herbert F. (1989) Primoridal electrical induction heating of asteroids. *Icarus*, 78, 402-410.

Herndon J. M. and Herndon M. A. (1977) Aluminum-26 as a planetoid heat source in the early solar system. *Meteoritics*, 12, 459-465.

Huss G. R., MacPherson G. J., Wasserburg G. J., Russell S. S., and Srinivasan G. (2001) Aluminum-26 in calcium-aluminum-rich inclusions and chondrules from unequilibrated ordinary chondrites. *Meteorit. Planet. Sci.,* 36, 975-997.

Keil K., Stoffler D., Love S. G., and Scott E. R. D. (1997) Constraints on the role of impact heating and melting in asteroids. *Meteorit. Planet. Sci.*, 32, 349-363.

Kita N. T., Nagahara H., Togashi S., and Morishita Y. (2000) A short duration of chondrule formation in the solar nebula: Evidence from ^{26}Al in Semarkona ferromagnesian chondrules. *Geochim. Cosmochim. Acta*, 64, 3913-3922.

Kuhi L. V. (1964) Mass Loss from T Tauri Stars. *Astrophysical J.*, vol. 140, p.1409.

LaTourette T. and Wasserburg G. J. (1997) Mg diffusion in anorthite: Implications for the formation of early solar system planetesimals. *Earth Planet. Sci. Lett.*, 158, 91-108.

Lee T., Papanastassiou D. A., and Wasserburg G. J. (1976) Demonstration of ^{26}Mg excess in Allende and evidence for ^{26}Al. *Geophys. Res. Lett.*, 3, 41-44.

Lee D. and Halliday A.(1999) Accretion Rates and the Tungsten and Lead Isotopic Compositions of the Earth and Moon. *29th LPSC*, abstract no. 1540.

London, K., Dongarra, J., Moore, S., Mucci, P., Seymour, K., Spencer, T. (2001a) End-user Tools for Application Performance Analysis, Using Hardware Counters, *International Conference on Parallel and Distributed Computing Systems, Dallas, TX, August 8-10, 2001.*

Dongarra, J., London, K., Moore, S., Mucci, P., Terpstra, D. (2001) Using PAPI for Hardware Performance Monitoring on Linux Systems. *Conference on Linux Clusters: The HPC Revolution, Urbana, Illinois, June 25-27, 2001.*

London, K., Moore, S., Mucci, P., Seymour, K., Luczak, R. (2001b) The PAPI Cross-Platform Interface to Hardware Performance Counters. *Department of Defense Users' Group Conference Proceedings , Biloxi, Mississippi, June 18-21, 2001.*

MacPherson G. J., Davis A. M., and Zinner E. K. (1995) The distribution of aluminum-26 in the early solar system – a reappraisal. *Meteorit. Planet. Sci.*, 30, 365-386.

McSween H. Y. Jr., Sears D. W. G., and Dodd R. T. (1988) Thermal metamorphism. In *Meteorites and the Early Solar System* (J. F. Kerridge and M. S. Matthews, eds.), pp. 102-113. University of Arizona Press, Tucson.

McSween H. Y. Jr,, Ghosh A., Grimm R. E., Wilson L. and Young E. (2003) Thermal Evolution Models of Asteroids. *Asteroid-III.* Univ. of Arizona Press, In Press.

Melosh H. J. (1990) Giant impacts and the thermal state of the early Earth. In *Origin of the Earth* (H. E. Newsom and J. H. Jones, eds.), pp. 69-84. Oxford University Press, New York.

Minster J. F. and Allegre C. J. (1979) ^{87}Rb-^{87}Sr chronology of H chondrites: Constraint and speculations on the early evolution of their parent body. *Earth Planet. Sci. Lett.,* 42, 333-347.

Miyamoto M. (1991) Thermal metamorphism of CI and CM carbonaceous chondrites: A internal heating model. *Meteoritics*, 26, 111-115.

Miyamoto M., Fujii N., and Takeda H. (1981) Ordinary chondrite parent body: An internal heating model. *Proc. Lunar Planet. Sci. Conf.* 12th, pp. 1145-1152.

NetSolve website (2003) http://icl.cs.utk.edu/netsolve

PAPI website (2003) http://icl.cs.utk.edu/papi

PAPI User Guide (2003) website:
http://icl.cs.utk.edu/projects/papi/files/documentation/PAPI_USER_GUIDE_23.pdf

Podosek F. and Cassen P. (1997) Theorerical, observational and isotopic estimates of the lifetime of the solar nebula. *Meteoritics,* 29, 6.

Rubin A. E. (1995) Petrologic evidence for collisional heating of chondritic asteroids. *Icarus*, 113, 156-167.

Shimazu H. and Terasawa T. (1995) Electromagnetic induction heating of meteorite parent bodies by the primordial solar wind. *J. Geophys. Res.*, 100, 16,923-16,930.

Sonnett C. P., Colburn D. S., and Schwartz K. (1968) Electrical heating of meteorite parent bodies and planets by dynamo induction from a premain sequence T Tauri "solar wind." *Nature*, 219, 924-926.

Srinivasan G., Goswami J. N., and Bhandari N. (1999) ^{26}Al in eucrite Piplia Kalan: Plausible heat source and formation chronology. *Science*, 284, 1348-1350.

Stoeffler D., Bischoff A., Buchwald V., Rubin A. E. (1988) Shock effects in meteorites. *Meteorites and the Early Solar System.* (Eds. Kerridge J. and Matthews J.) Univ. of Arizona Press, Tucson.

Urey H. (1955) The cosmic abundances of potassium, uranium, and thorium and the heat balances of the Earth, the Moon and Mars. *Proc. Natl. Acad. Sci. U.S.*, 41, 127-144.

User's Guide to Netsolve V1.4 by Dorian Arnold, Sudesh Agrawal, Susan Blackford, Jack Dongarra, Michelle Miller, Sathish Vahdiyar, Kiran Sagi, Zhiao Shi. University of Tennessee, CS Department Techincal Report No. UT-CS-01-467 July, 2001.

Weidenschilling S. J., Spaute D., Davis D. R., Mazari F., and Ohtsuki K. (1997) Accretional evolution of a planetesimal swarm. *Icarus*,128, 429-455.

Wetherill G. W. (1980) Formation of terrestrial planets. *Ann. Rev. Astron. Astrophys.*,18, 77-213.

Wilson L. and Keil K. (2000) Crust development on differentiated asteroids. In *Lunar and Planetary Science XXXI*, CD #1576. Lunar and Planetary Institute, Houston.

Wilson L., Keil K., Browning L. B., Krot A. N., and Bourcher W. (1999) Early aqueous alteration, explosive disruption, and reprocessing of asteroids. *Meteorit. Planet. Sci*., 34, 541-557.

Wood J. A. (1979) Review of the metallographic cooling rates of meteorites and a new model for the planetesimals in which they formed. In *Asteroids* (T. Gehrels, ed.), pp. 849-891. University of Arizona Press, Tucson.

Yomogida K. and Matsui T. (1984) Multiple parent bodies of ordinary chondrites. *Earth Planet. Sci. Lett.*, 68, 34-42.

Young E. D., Ash R. D., England P., and Rumble D. III (1999) Fluid flow in chondrite parent bodies: deciphering the compositions of planetesimals. *Science,* 286, 1331-1335.

Appendix

Table 1A: Chronological summary of published asteroid thermal evolution models

Model	Description
Urey (1955)	First feasibility calculation of ^{26}Al as an asteroid heat source
Sonnett et al. (1968)	First proposal for electromagnetic induction heating of asteroids
Herndon and Herndon (1977)	Feasibility study of ^{26}Al as an asteroid heat source
Fujii et al. (1979)	Comparison of internal and external heating models for asteroids
Minster and Allegre (1979)	^{26}Al heating model for the H-chondrite parent body
Wood (1979)	Model to reproduce metallographic cooling rates of iron meteorites
Miyamoto et al. (1981)	^{26}Al heating model to constraine sizes of Oc parent bodies using cooling rates, isotopic closure ages, and fall statistics
Yomogida and Matsui (1984)	^{26}Al heating model for small, unsintered asteroids
Grimm (1985)	Model of asteroid metamorphism with fragmentation and reassembly
Grimm and McSween (1989)	^{26}Al heating model of ice-bearing planetesimals, to account for aqueous alteration in Cc
Herbert (1989)	Model of electromagenetic induction heating which causes melting
Haack et al. (1990)	Thermal model of a differentiated asteroid based on decay of long-lived radionuclides
Miyamoto (1991)	^{26}Al heating model to account for aqueous alteration in Cc asteroids
Grimm and McSween (1993)	Explanation of inferred thermal stratification of the asteroid belt based on heliocentric accretion and ^{26}Al heating
Shimazu and Terasawa (1995)	Model of electromagnetic induction heating
Bennett and McSween (1996)	Updated ^{26}Al heating model for Oc asteroids, using revised chronology and thermophysical properties
Akridge et al. (1998)	Model for ^{26}Al heating of 6 Hebe with a megaregolith
Ghosh and McSween (1998)	^{26}Al heating model of HED parent body 4 Vesta
Wilson et al. (1999)	Overpressure and explosion resulting from heating Cc asteroids
Young et al. (1999)	^{26}Al heating model of Cc asteroids with fluid flow, to explain oxygen isotope fractionations
Cohen and Coker (2000)	Short- and long-lived radionuclide heating model of Cc parent bodies used to study racemization of amino acids
Wilson and Keil (2000)	Thermal effects of magma migration in 4 Vesta
Ghosh et al. (2001)	Effect of incremental accretion on inferred thermal distribution of asteroids in the main belt

Table 2A: Verification of Numerical Error by Adjusting Coarseness of the Mesh

No. of elements	T	Est. Error	Est. T(exact)	Slope
15	946.40			
30	940.57	1.94	938.63	
60	939.16	0.47	938.69	2.05

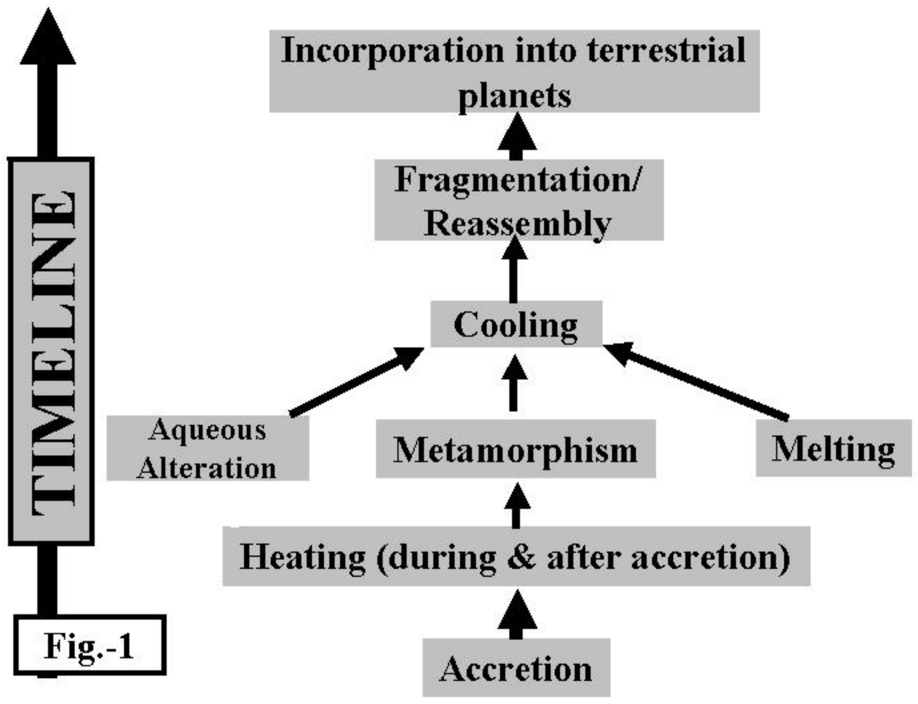
Incorporation into terrestrial planets
Fragmentation/ Reassembly
Cooling
Aqueous Alteration
Metamorphism
Melting
Heating (during & after accretion)
Accretion
TIMELINE
Fig.-1

Figure A1: Schematic diagram summarizing asteroidal evolution. Note that a generic asteroid might not go through all the stages. Also, current state of knowledge cannot distinguish whether some stages were sequential or partly contemporaneous, e.g. it is likely that heating took place during as well as after the period of accretion was over. Also, the relative timing of certain stages might be different. For example, fragmentation and reassembly could have taken place during or after cooling.

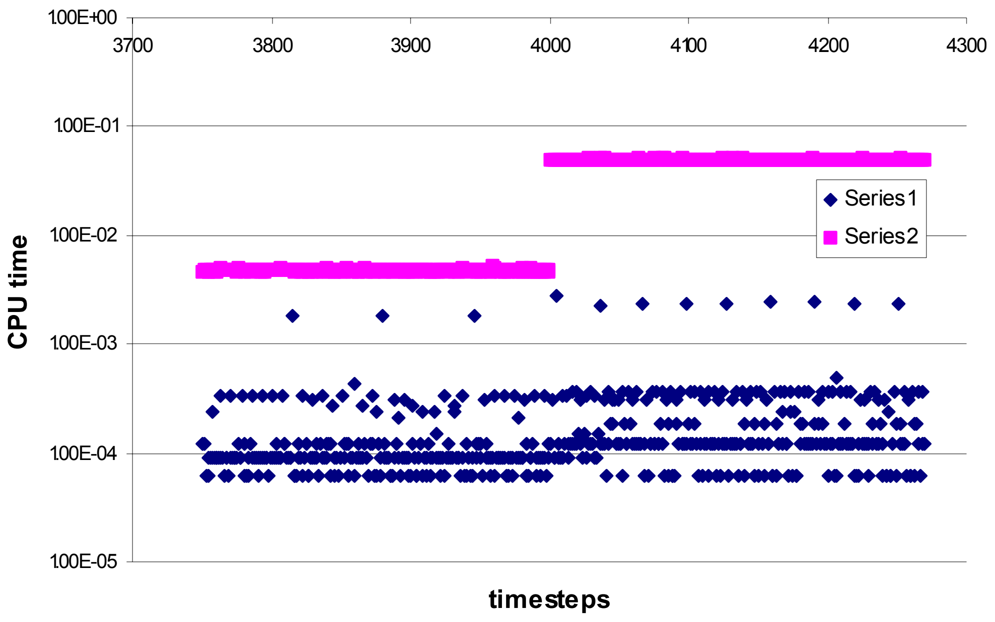

Figure A2: Comparison of CPU time between Gauss elimination and the tridiagonal solver for matrix sizes of 46 and 99. The X-axis plots the number of timesteps for the heat transfer code. The graph shows approximately 250 timesteps in each of the two domains. The Y-axis shows the CPU time in logarithmic scale. For approximately, the left half portion of the graph, the code is in time domain-1, where the size of A = [46,46]. and on the right half the code is in domain-2 where the matrix size is [99,99]. Series 1 represents the CPU time for the new routine (blue in color). Series 2 represents the CPU time for the old Gauss elimination routine (pink in color). The diagram shows that CPU time used for Gauss elimination increases by about *one* order of magnitude as the size of A increases from (46X46) to (99X99). In case of the tridiagonal solver, the increase is at best insignificant, given the logarithmic scale of the Y-axis.

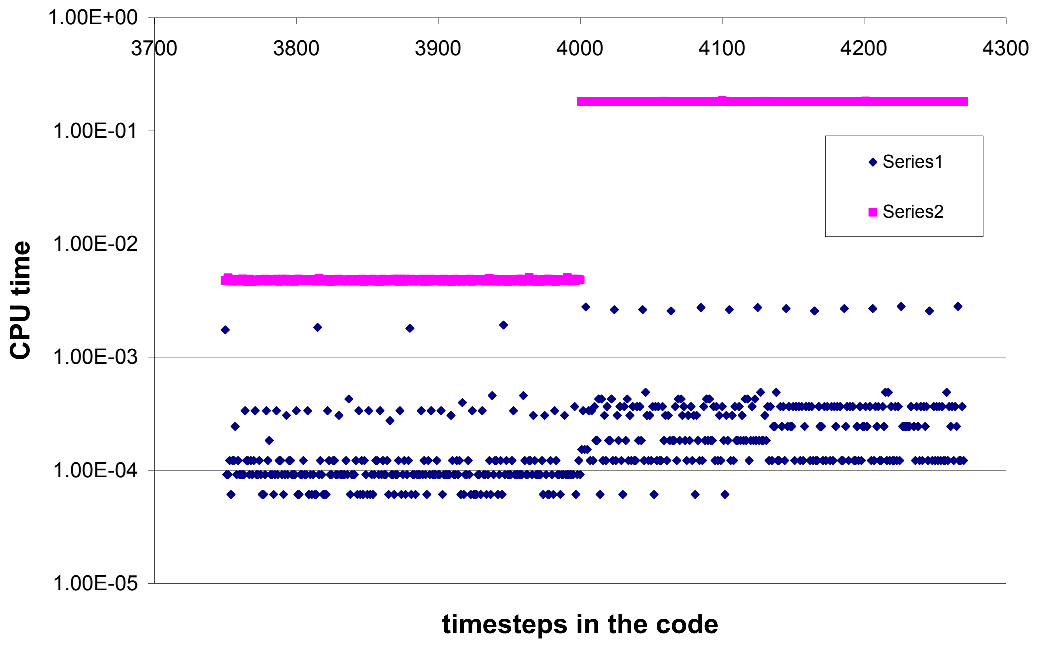

Figure A3: Comparison of CPU time between Gauss elimination and the tridiagonal solver for matrix sizes of 46 and 151. The X-axis plots the number of timesteps for the heat transfer code. The Y-axis shows the CPU time in logarithmic scale. The matrix size on the left half of the plot is [46,46] and on the right half is [151,151]. Series 1 represents the CPU time for the new routine (blue in color). Series 2 represents the CPU time for the old Gauss elimination routine (pink in color). The diagram shows that CPU time used for Gauss elimination increases by more than an order of magnitude as the size of A increases from (46X46) to (151X151). In case, of the tridiagonal solver, the increase in CPU time is insignificant.

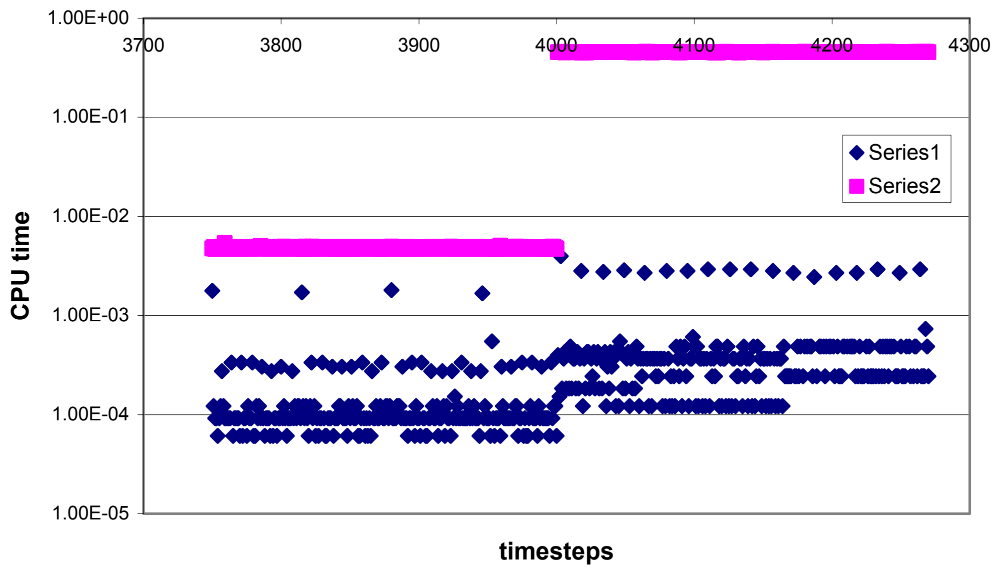

Figure A4: Comparison of CPU time between Gauss elimination and the tridiagonal solver for matrix sizes of 46 and 199. The X-axis plots the number of timesteps for the heat transfer code. The Y-axis shows the CPU time in logarithmic scale. The left half of the plot is [46,46] and on the right half is [199,199]. Series 1 represents the CPU time for the new routine (blue in color). Series 2 represents the CPU time for the old Gauss elimination routine (pink in color). The diagram shows that CPU time used for Gauss elimination increases by about *two* orders of magnitude as the size of A increases from (46X46) to (199X199). In case, of the new solver, the increase in CPU time is insignificant.

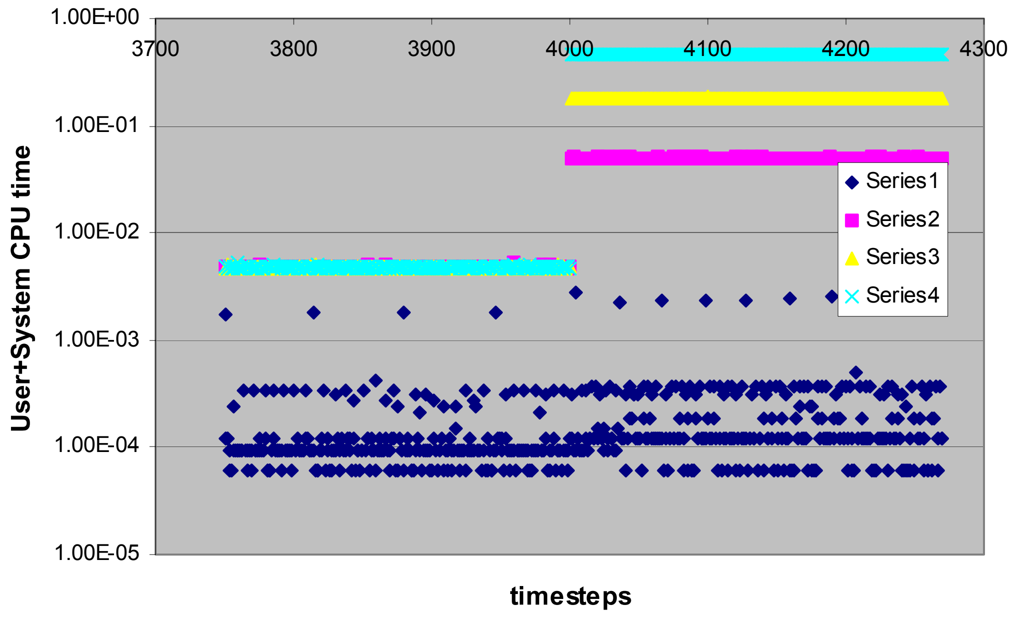

Figure A5: Summary of the variation of CPU time with matrix size for the Gauss elimination and tridiagonal solver routines. The X-axis plots the last 250 timesteps of time domain-1, followed by the first 250 timesteps of time domain-2. The Y-axis plots CPU time in seconds on a logarithmic scale. As n (where, size of A is (n,n)) increases from 46 to 199, the CPU time for Gauss elimination increases almost by two orders of magnitude. Series 1 (blue) represents the tridiagonal solver. Series 2 (pink), Series 3 (yellow) and Series 4 (light blue) represent outputs for Gauss elimination where the size of the matrix in time domain 2 are 99, 149, 199, respectively. In time domain 1, Series-2, 3 and 4 have a matrix size of 46. Since Gauss elimination is $O(n^3)$, increasing n from 46 to 199 (>4 times) should cause the time to increase by $199/46=4.32^3= 81$. CPU time for the new solver on the other hand is about a order of magnitude lower than that of Gauss elimination for n=46. More importantly, the increase in CPU time is not noticeable for the sparse matrix solver since the solver is $O(n)$, whereas Gauss elimination is $O(n^3)$.

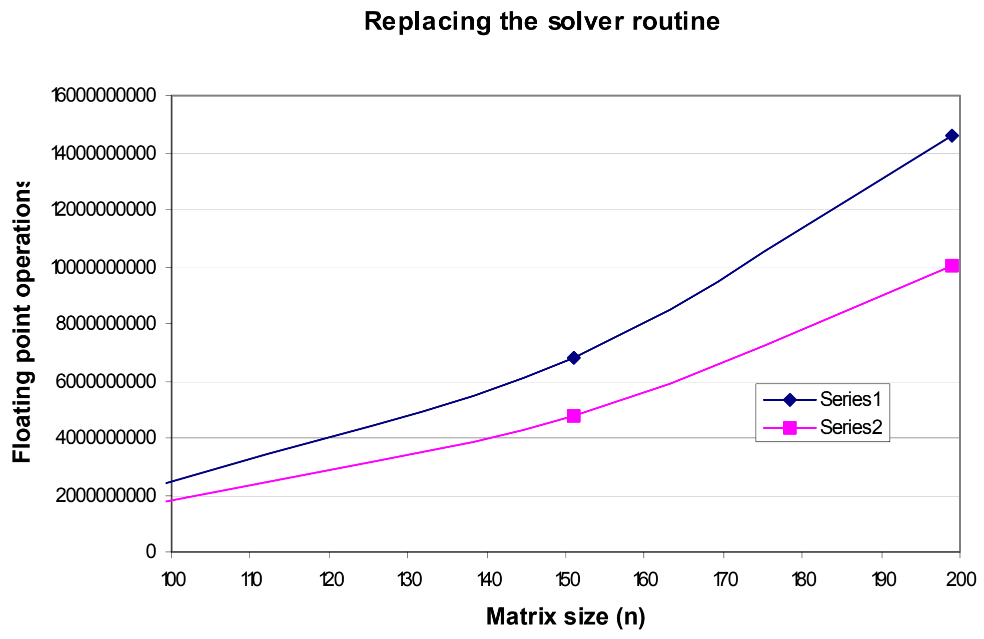

Figure A6: Comparison of the number of floating point operations for the (entire) code with the Gauss elimination and tridiagonal solver routines, respectively. Series-1 (blue) represents the code with the Gauss elimination, whereas Series-2 (pink) represents the code with the tridiagonal solver. Since the sparse matrix solver is *O*(n), a larger percent difference is observed at higher matrix sizes compared to Gauss elimination.

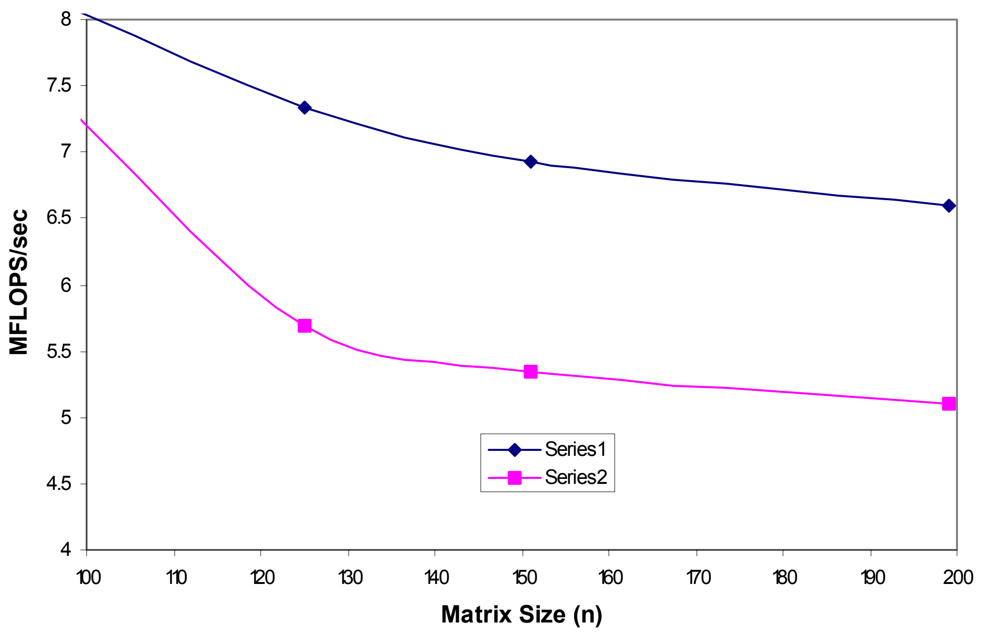

Figure A7: Summary of performance (in MFLOPS/s) versus matrix size for the (entire) code with the Gauss elimination solver and the Sparse Matrix solver as a function of matrix size. Series-1 (blue) represents the code with the Gauss elimination, whereas Series-2 (pink) represents the code with the tridiagonal solver. A slight degradation in performance of ~2 MFLOPS/s is observed. This can be attributed to the sparse tridiagonal solver used in place of the Gauss elimination routine. The algorithm for the tridiagonal solver decreases the number of flops, since it produces operations that are dependent on previous set of operations. Thus, one iteration of the do loop cannot initiate until the last iteration is completed, since the present iteration uses a variable calculated in the last iteration. As a result, the code is strictly serial and cannot be vectorized. The Gauss elimination algorithm has considerable independent floating point operations that can be executed independently. Thus, the code can be vectorized to some extent.

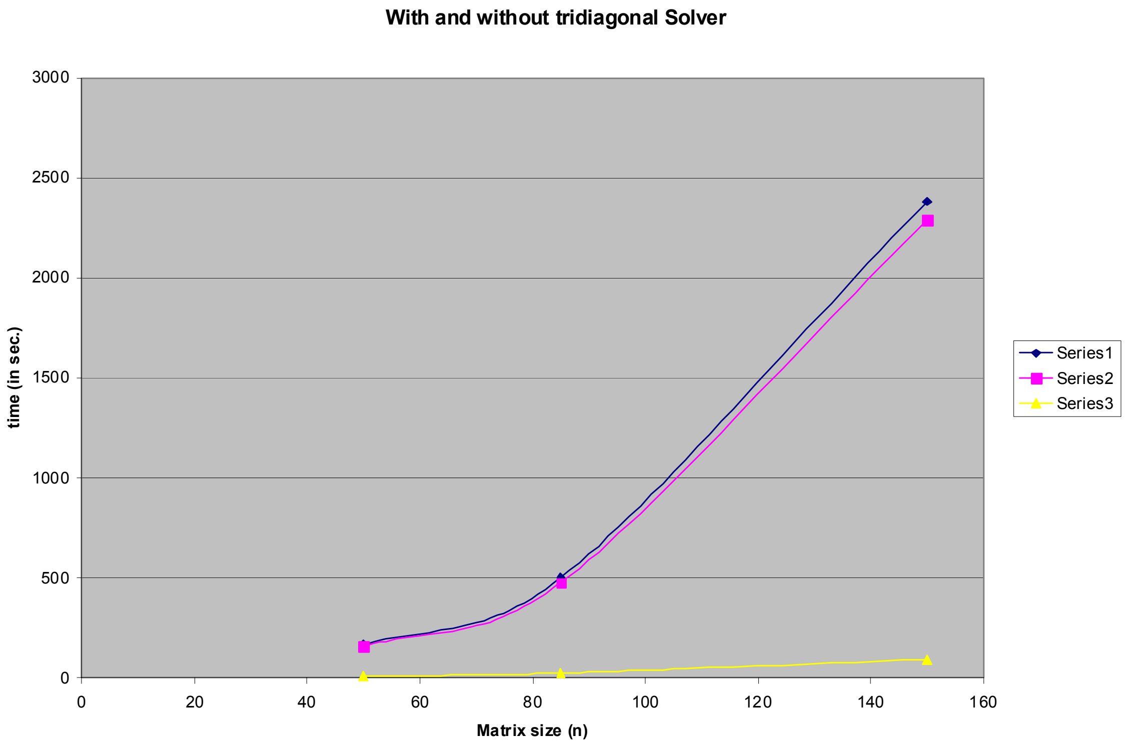

Figure A8: Comparison of run times of the original code (with the Gauss elimination solver) and the code with the tridiagonal solver. The X-axis and Y-axis plot matrix size and CPU time in seconds, respectively. The blue line (series 1) signifies the run time in the original code, the pink line (series 2) indicates the run time using the tridiagonal solver in place of Gauss elimination. The yellow line (series 3) calculates the difference in run time or the improvement of series 2 over series 1. As is clearly apparent, the solver routine causes a reduction in run time for the entire code, but the improvement is insignificant.

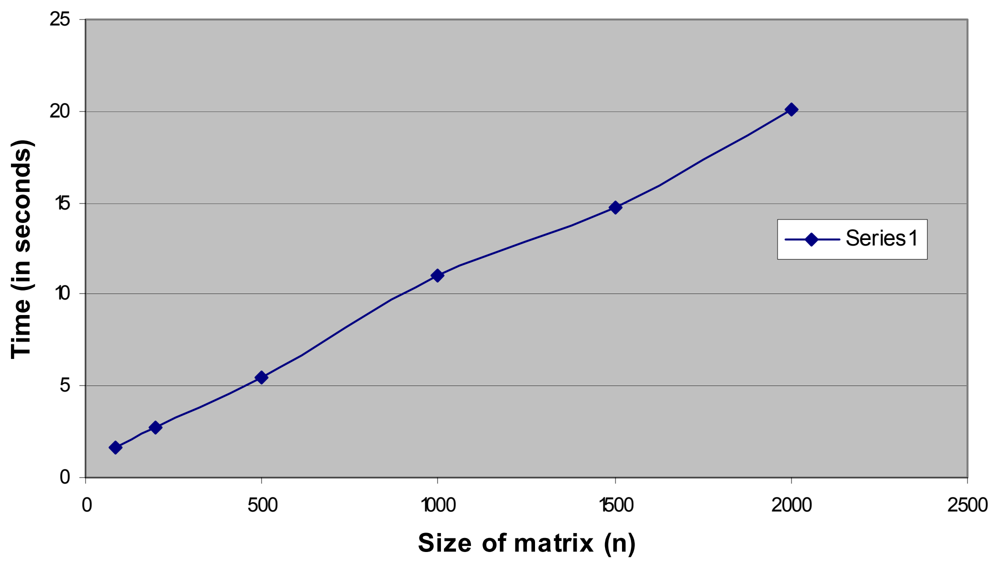

Figure A9: The performance of the code with all finite element matrices reduced to their corresponding sparse matrix form. The X-axis and Y-axis plot matrix size and CPU time in seconds, respectively. For a matrix size of 85, the run time decreases from 499 seconds to 1.66 seconds: a decrease of 99.7%. Note that the sparse matrix optimization frees up memory in the code and enables the maximum matrix size to increase from about 200 to 2000. As expected if the operations were *O*(n), the increase in matrix size causes almost a linear increase in time.

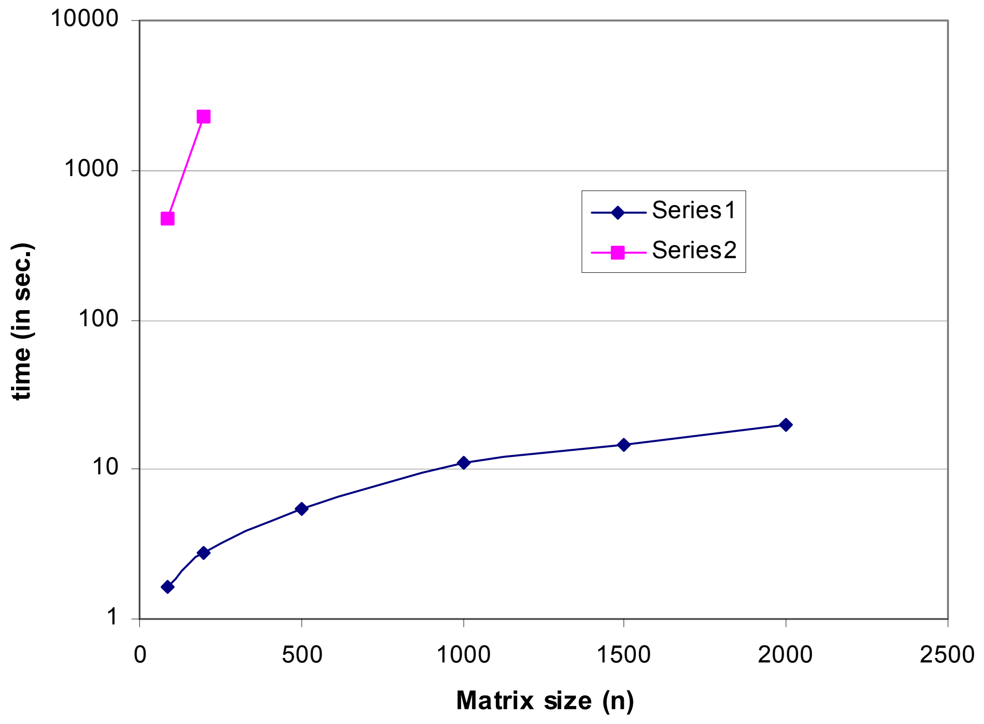

Figure A10: Comparison of the run time for code before and after reducing all finite element matrices to the sparse matrix form. The X-axis and Y-axis plot matrix size and the number of floating point operations, respectively. The X-axis and Y-axis plot matrix size and CPU time in seconds, respectively. The plot above summarizes the difference in time between the previous version of the code (with the sparse solver: Series-2) and the present version (sparse solver and sparse matrix representation of finite element matrices: Series-1). Note that the scale for the Y-axis is logarithmic. Also, note that the change of run time for Series-1 is far more gradual, whereas for Series-2 is very steep. In the former case, the code is O(n) and in the latter case, the code is $O(n^3)$.

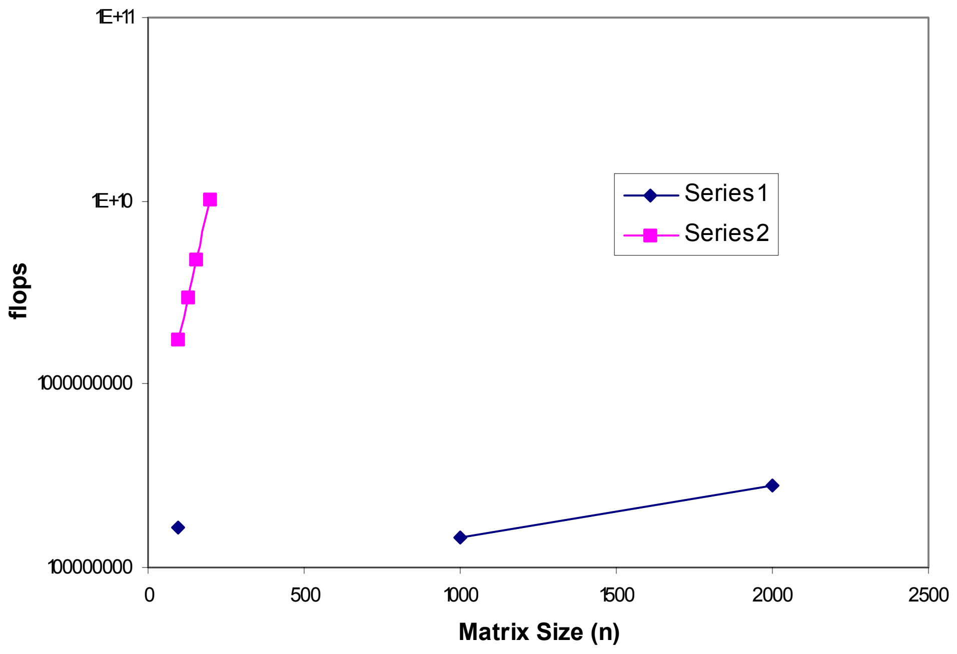

Figure A11: Comparison of floating point operations for code before and after reducing all finite element matrices to the sparse matrix form. The X-axis and Y-axis plot matrix size and CPU time in seconds, respectively. Series-1 (blue) represents the optimized code whereas Series-2 (pink) represents the code from the previous iteration. For a matrix size of 100, the optimized code reduces the number of floating point operations by an order of magnitude. For larger matrix sizes, because the optimized code is *O*(n) and the unoptimized code was *O*(n^3), the difference in the number of flops between the optimized and unoptimized code will be much larger.

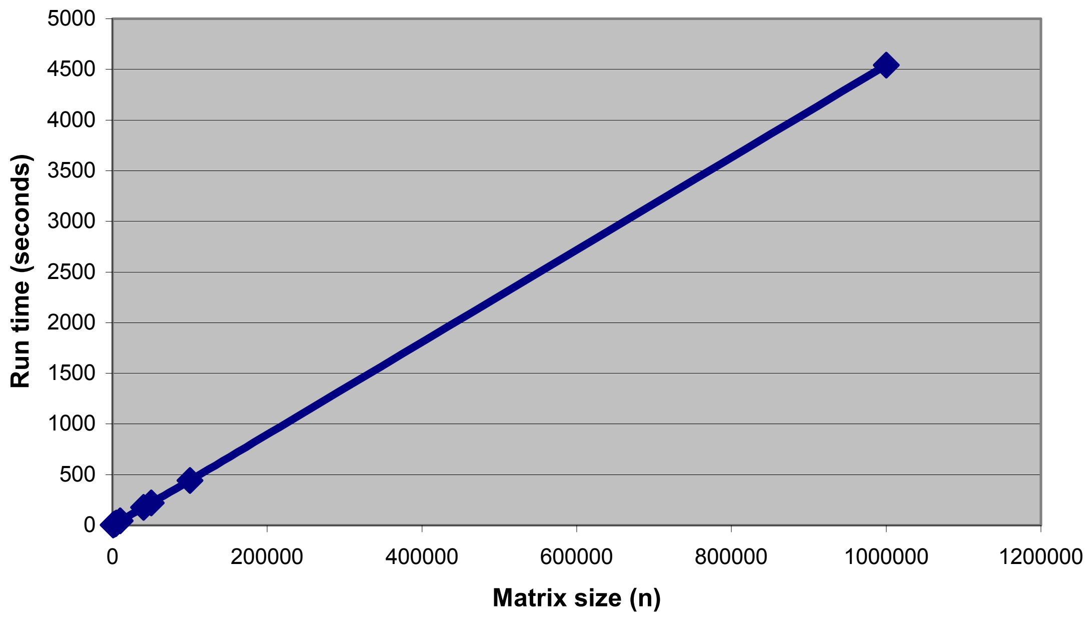

Figure A12: Plot of run time with matrix size after all finite element matrices were expressed in the sparse matrix form, and after all 2D and 3D matrices are expressed as a 1D arrays. The X-axis and Y-axis plot matrix size and CPU time in seconds, respectively. Thus, after this iteration all matrices are expressed as one-dimensional arrays is sparse matrix form, where the diagonal and lower diagonal are stored as arrays of size n and n-1, respectively. The run time of the code was reduced by 66%: thus, for a problem size of 1000, the run time was reduced from ~ 12 seconds to ~4 seconds. Significantly, the maximum problem size that could be run on Cetus increased from 2000 to 5,000,000, i.e. 2500 times. The problem is *O*(n) and thus the increase in matrix size causes almost a linear increase in time.

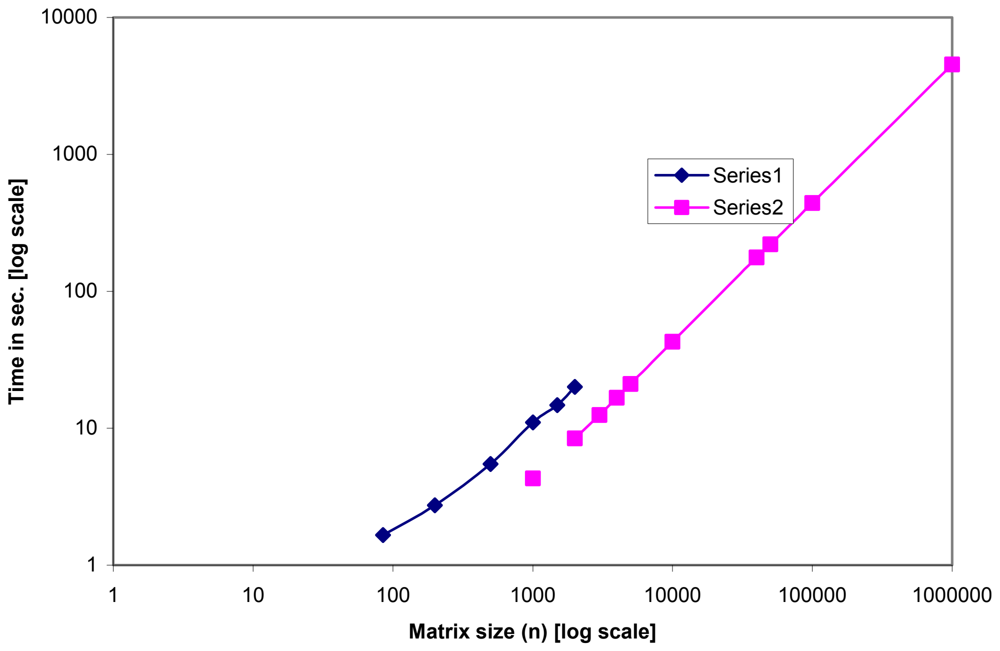

Figure A13: Comparison of run time with matrix size after all finite element matrices were expressed in the sparse matrix form, and after all 2D and 3D matrices are expressed as a 1D array, respectively. The figure shows matrix size versus time for the present version (Series-2: pink in color) compares to the previous version (Series-1: blue in color). Note that scales on the X- and Y-axis are logarithmic. Note that Series-2 (present version) is offset to the left of Series-1 (previous version). This means that run time for the same problem size is lower for the optimized version. For the previous version, runs could be made for matrix sizes of upto 2000. For the present version, matrix sizes can be increased upto 1,000,000.

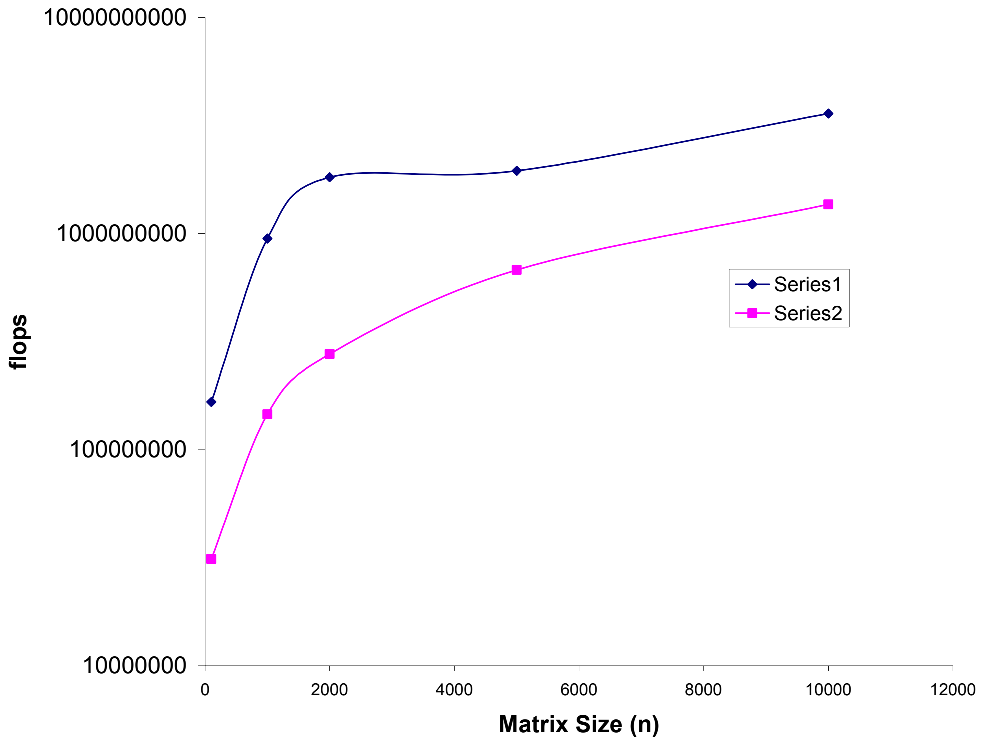

Figure A14: Comparison of the number of floating point operations with matrix size after all finite element matrices were expressed in the sparse matrix form, and after all 2D and 3D matrices are expressed as a 1D array, respectively. The X-axis and Y-axis plot matrix size and the number of floating point operations per second, respectively. The figure summarizes the decrease in the number of floating point operations between the present version (Series-2: pink in color) and the previous version (Series-1: blue in color). Note that the Y-axis is logarithmic. Thus, at matrix size 1000, the flops for the previous version is 9.49E+08. For the present version, the number of flops is 1.46E+08: thus the present optimization reduces the number of flops by 85%.

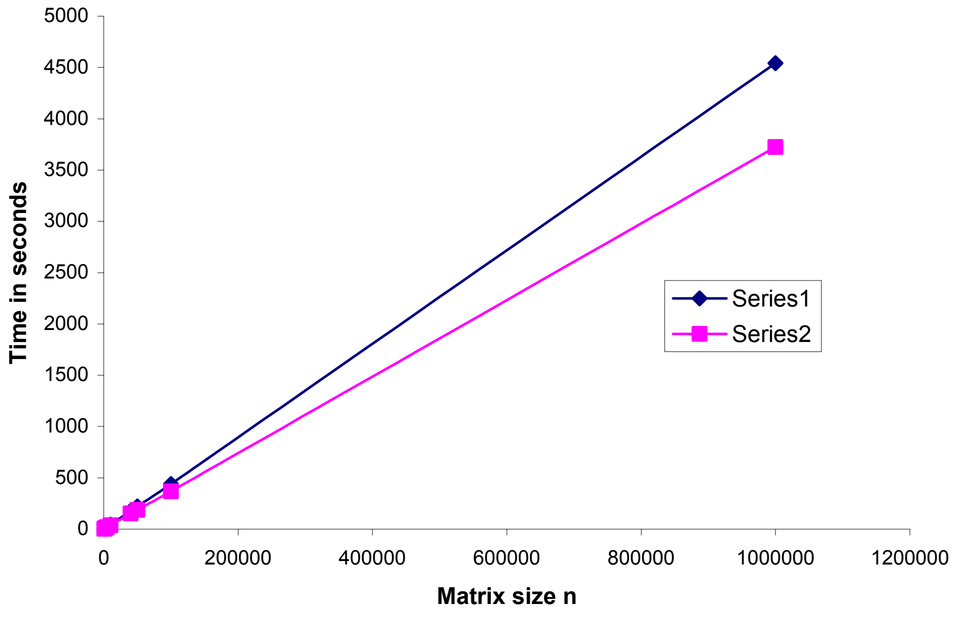

Figure A15: Comparison of run time for the code before and after optimizing to reduce the number of vector touches and merging of do loops to enable cache reuse. The X-axis and Y-axis plot matrix size and CPU time in seconds, respectively. The optimized code is represented by Series-2 (pink in color), whereas the previous version of the code is represented by Series-1 (blue in color). The improvement in time is ~8% at matrix sizes of 1000, and 18% at matrix sizes of 1,000,0000.

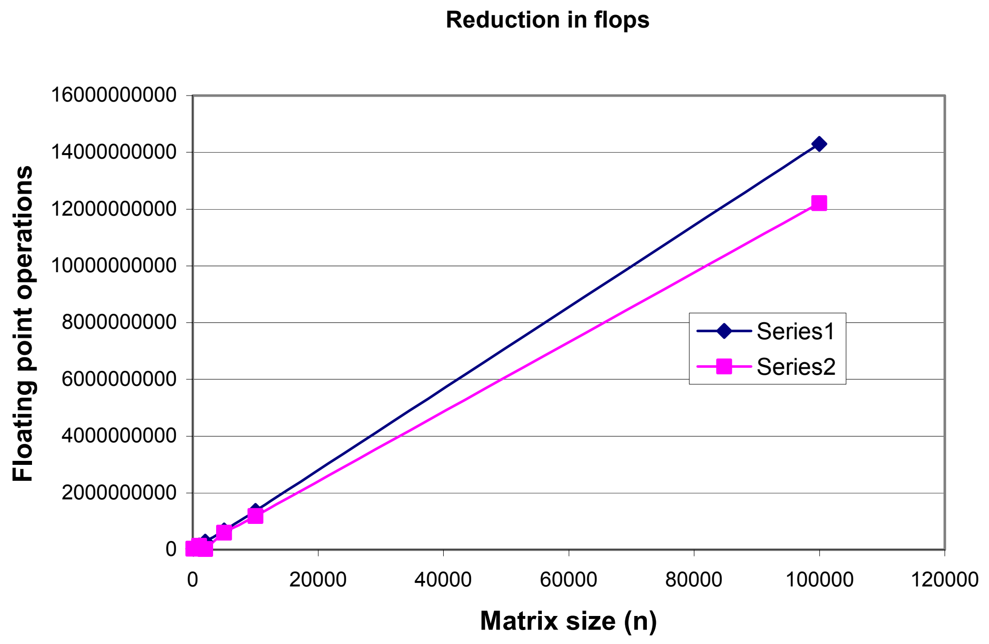

Figure A16: Comparison of the number of floating point operations for the code before and after optimizing to reduce the number of vector touches and merging of do loops to enable cache reuse. The X-axis and Y-axis plot matrix size and the number of floating point operations, respectively. The optimized code is represented by Series-2 (pink in color), whereas the previous version of the code is represented by Series-1 (blue in color). The number of flops is observed decrease between 4 – 14%.

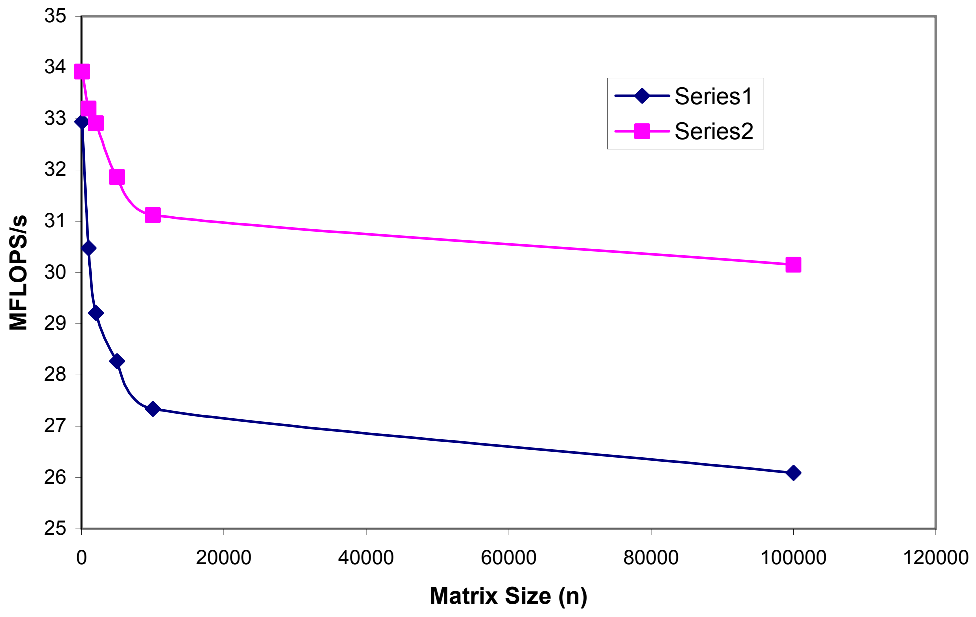

Figure A17: Comparison of performance (in MFLOPS/s.) for the code before and after optimizing to reduce the number of vector touches and merging of do loops to enable cache reuse. The X-axis and Y-axis plot matrix size and performance in MFLOPS/second, respectively. The optimized code is represented by Series-1 (blue in color), whereas the previous version of the code is represented by Series-2 (pink in color). There is a marginal improvement in the speed measured in MFLOPS/s between 3 (at matrix size=100) – 15% (at matrix size = 100000).

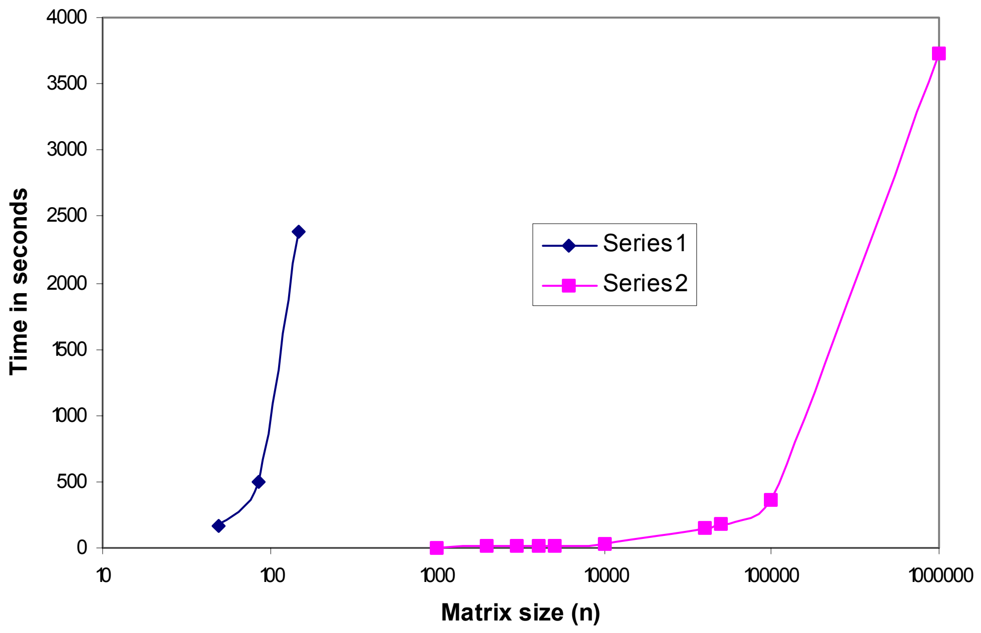

Figure A18: Comparison of the run time of the original code and the final optimized version. The X-axis and Y-axis plot matrix size and CPU time, respectively. Note that the X-axis is logarithmic. The original code is represented by Series-1 (blue in color), while the optimized version is represented by Series-2 (pink in color). The optimization reduced the CPU time taken to run the code from 297 sec to 0.88 sec for a matrix size of 100, an improvement of 99.70%. More importantly, the algorithm was reduced from a $O(n^3)$ operation to a $O(n)$ operation. Thus, the percent time difference between the optimized and unoptimized versions will be greater at larger matrix sizes.

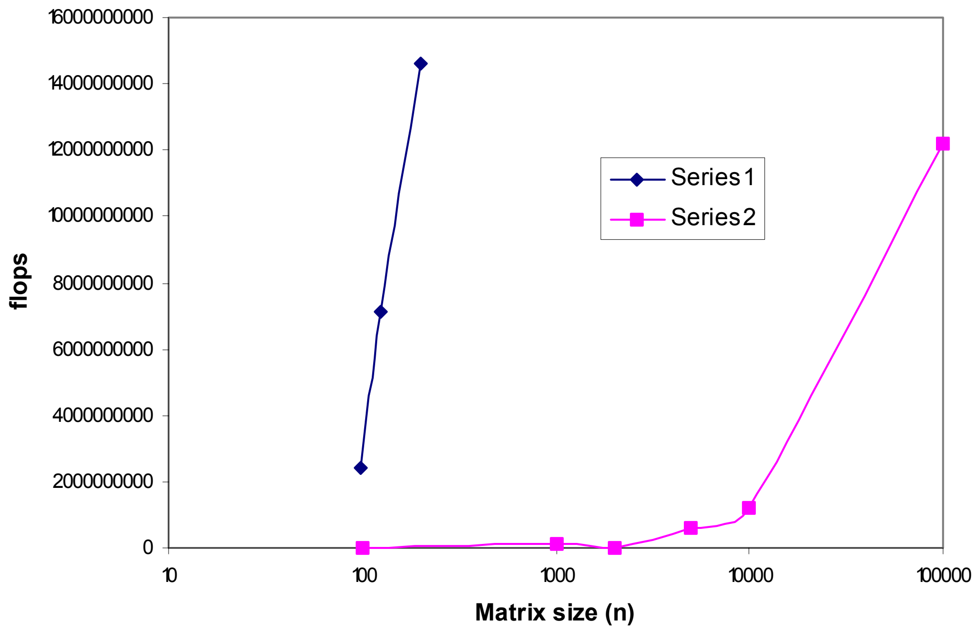

Figure A19: Comparison of the number of floating point operations of the original code and the final optimized version. The X-axis and Y-axis plot matrix size and the number of floating point operations, respectively. Note that the X-axis is logarithmic. The original code is represented by Series-1 (blue in color), while the optimized version is represented by Series-2 (pink in color). At matrix sizes of 100, the number of floating point operations were reduced from 2.39 E+09 to 2.99E+07, an improvement of 98.75%. As, the algorithm is reduced from a $O(n^3)$ operation to a $O(n)$ operation; the difference between the optimized and unoptimized versions is even greater at larger matrix sizes.

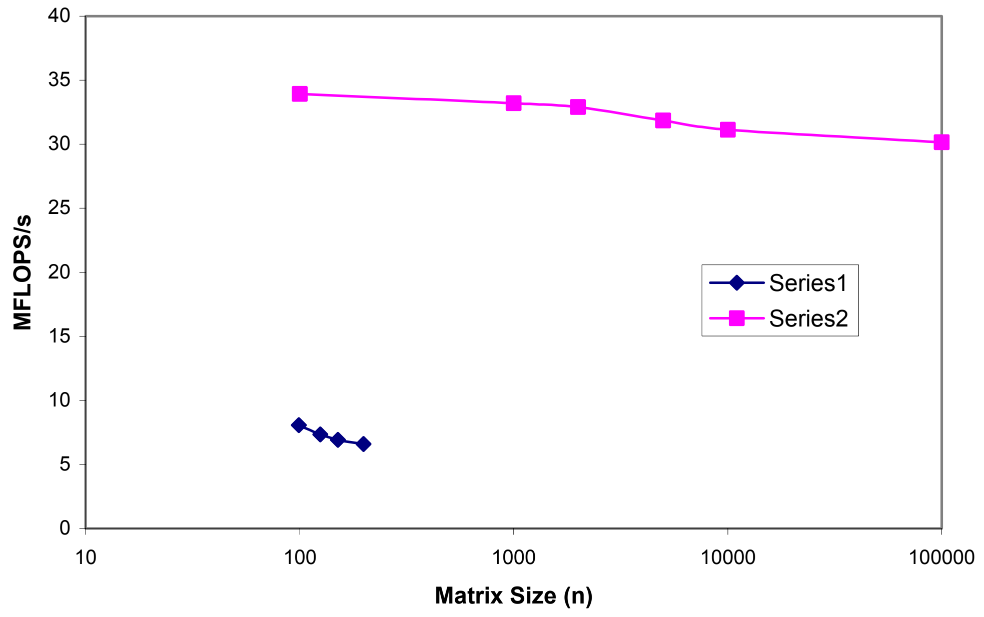

Figure A20: Comparison of performance in MFLOPS/s. of the original code and the final optimized version. The X-axis and Y-axis plot matrix size and performance in MFLOPS/second, respectively. Note that the X-axis is logarithmic. The original code is represented by Series-1 (blue in color), while the optimized version is represented by Series-2 (pink in color). At matrix sizes of 100, the performance increased by about 4 times, from 8.06 MFLOPS/s to 33.92 MFLOPS/s

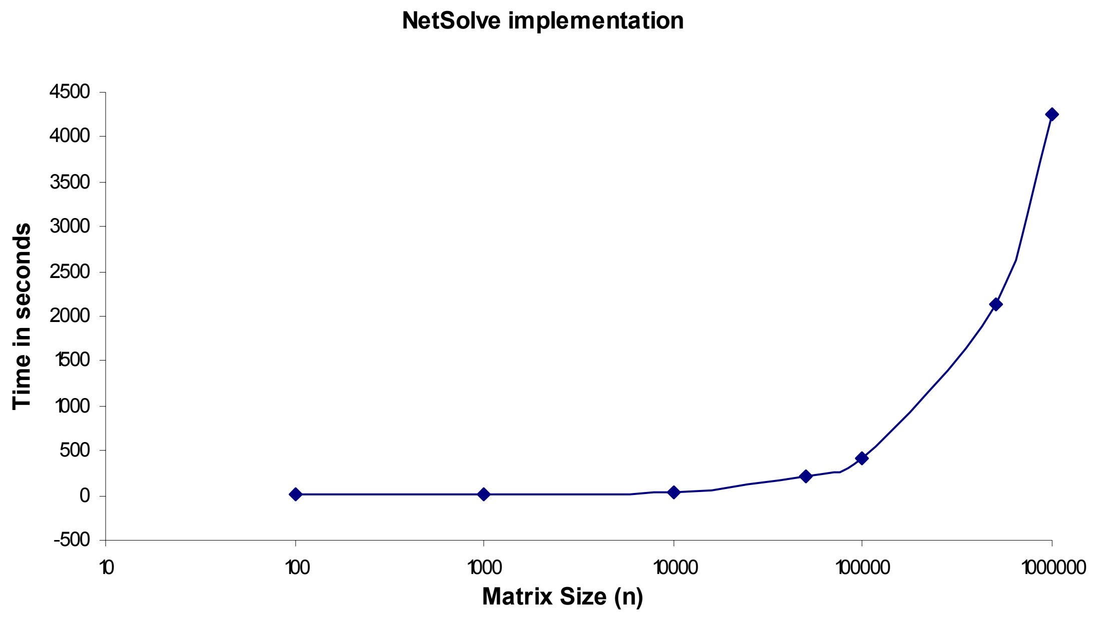

Figure A21: Run time versus matrix size for NetSolve implementation of the code. The X-axis and Y-axis plot matrix size and CPU time in seconds, respectively. The results as a function of matrix size and time are summarized in the figure below. Note that the X-axis is logarithmic.

VITA

Amitabha Ghosh was born in Calcutta, India in 1970. He completed his B.Sc. in Geological Sciences and M.Sc. in Applied Geology from the Indian Institute of Technology, Kharagpur, India, in 1991 and 1993, respectively. He completed his Ph.D. degree in Geological Sciences from the University of Tennessee in 1997. In July 1997, he served in Mars Pathfinder Mission Operations as part of the Mineralogy Science Operation Group. As member of the Mars Pathfinder Mission, he conducted chemical analysis of rocks and soil in the landing site. He analyzed the first ever-Martian rock, and was the recipient of the NASA Mars Pathfinder Group Achievement Award for work during the mission. He is currently working on data analysis for THEMIS instrument on the Mars Odyssey Mission (2001 – 2005).