HIGH PERFORMANCE ALGORITHMS TO IMPROVE THE RUNTIME COMPUTATION OF SPACECRAFT TRAJECTORIES

A Thesis
Presented to
The Academic Faculty

by

Nitin Arora

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
Guggenheim School of Aerospace Engineering

Georgia Institute of Technology
August 2013

HIGH PERFORMANCE ALGORITHMS TO IMPROVE THE RUNTIME COMPUTATION OF SPACECRAFT TRAJECTORIES

Approved by:

Dr. Ryan P. Russell, Advisor
The Department of Aerospace
Engineering and Engineering
Mechanics
The University of Texas at Austin

Dr. Robert D. Braun (co-advisor)
Guggenheim School of Aerospace
Engineering
Georgia Institute of Technology

Dr. Marcus J. Holzinger
Guggenheim School of Aerospace
Engineering
Georgia Institute of Technology

Dr. Richard Vuduc
School of Computational Science and
Engineering
Georgia Institute of Technology

Dr. P.K Yeung
Guggenheim School of Aerospace
Engineering
Georgia Institute of Technology

Date Approved: June-27-2013

"I can live with doubt and uncertainty. I think it's much more interesting to live not knowing than to have answers which might be wrong."

Richard P. Feynman

Dedicated to mom and dad

who always believed in me

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor *Dr. Ryan Russell* for his understanding and support throughout my graduate studies. Thank you, *Ryan* for giving me an opportunity to work with you. Your patience, ingenuity, clear vision, and positive criticism have enriched my experience as a graduate student. Thank you for putting up with me even when I wasnt the easiest student to advice.

I would also like to acknowledge my committee members for their invaluable inputs and support over the course of my research. *Dr. Robert Braun*, apart from your wise and invaluable research advice, I would like to thank you for helping me stay in the lab as your student towards the end of my graduate studies. *Dr. Rich Vuduc*, I am really grateful to you for introducing me to the world of high performance computing and allowing me to be a part of your lab. I enjoyed our quick afternoon meetings, where I always ended up learning something new. *Dr. Marcus Holzinger*, thank you for allowing me to be your teaching assistant and for your valuable advisement while being a part of my research committee. *Dr. P.K Yeung*, I am very grateful for your unique perspective on my research and for your valuable inputs as my committee member. I would also like to acknowledge the graduate coordinator of the School of Aerospace Engineering, *Dr. Jeff Jagoda*. Your continuous support has helped me sail through many troubled waters, thanks a lot!

Being part of the Space Systems Design Lab and school of Aerospace Engineering at Georgia Tech has left me with a lot of fond memories. From qualifiers to my thesis defense, I have always enjoyed the continuous support and constructive criticism from my fellow labmates and friends. Particularly, I would like to thank *Gregory Lantoine, Brad Steinfeldt, Richard Otero, Jean-Francois Castet, Nuno Filipe, Francesca Favaro,*

Som Dutta, Lin Li, Veronica Foreman and *Cengiz Akinli* for providing me the much-needed support at Georgia Tech. I cannot forget my fellow labmates at the University of Texas at Austin. Thanks a lot *Vivek Vittaldev, Demyan Lantukh* and *Etienne Pellegrini* for putting up with me. Best of luck to you all!

I would also like to take this opportunity to acknowledge NASA, Air Force Research Lab and Emergent for supporting the work presented in this thesis. I would also like to acknowledge some of the people at the Jet Propulsion Laboratory who made my internship experience really special and helped me gain confidence as a researcher. Thanks a lot *Nathan J. Strange, Farah Alibay, Julim Lee, Francesco Simeoni, Jeffery Stuart, Damon Landau, Anastassios E. Petropoulos, Gregory J. Whiffen, Daniel Grebow, Jon A. Sims, Antranik Kolanjian* and *Shyam Bhaskaran.*

I would like to thank all my close friends back in India and here in the US. I have known some for more than 10 years while there are others that I have met during my time at Georgia Tech. I would like to especially thank *Ankur Bajoria, Rishab Mahajan, Abhijit Kamra, Akaash Jain, Nikhil Nanda, Shashank Shekhar Singhal, Amit Sharma, Saurabh Nagpal, Rajan Arora, Siddharth Sharma* and *Aditya Bhatt* for bearing with me for past 5+ years. All of you are absolutely amazing! *Prabuddha Bansal*, thanks a lot for the competitive and fun-filled squash games we played over the past five years. *Divya Bhatia*, I have enjoyed all the arguments and our various discussion about the universe. Thanks for putting up with me. *Kiran Girdhar*, thanks a lot for always being there for me. I really enjoyed your impromptu weekend trips to Atlanta. *Shaloo Rakheja*, I really cherished the time we spent together at Georgia Tech. Saying that we have been through a lot together is an understatement. Thanks a lot for you care, support and the continuous reminders about my terrible English.

I would also like to acknowledge the Star Trek franchise for the wonder, joy, and motivation it provided me during those long working hours.

Finally, I would give like to give a special thank you to my *Mom, Dad* and my

brother. Your unyielding love and support has always made me feel special and helped me rise above my best. *Mom* and *Dad*, thanks for believing in my dreams. You are the best! *Rohan*, thanks for being the best younger brother, you rock!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Increasing space mission complexity coupled with challenging science requirements are driving the need for fast and robust space trajectory design and simulation tools. Current state-of-the art methods and techniques are often found to be lacking, particularly when problems are scaled to the future demands of mission design.

This challenging problem is addressed in this thesis by 1) identifying a set of high impact "building-block" astrodynamics algorithms, 2) systematically improving several current state-of-the art solution methods via theoretical and methodological improvements and 3) taking advantage of modern computational hardware and numerical techniques to provide significant improvements in speed and robustness. In this thesis, five high impact astrodynamics problems are identified and their algorithms are selected for improvement. The solutions to the selected problems have applications ranging from preliminary mission design to high-fidelity space trajectory design and simulation.

The first problem identified is the multiple-revolution Lambert problem. Lambert's problem is one of the most extensively studied problems in space-flight mechanics and enjoys a large volume of research, spanning over several decades. In this thesis, a new formulation of the multiple revolution Lambert problem is presented. The formulation is based on a cosine transformation and uses rational functions for generating accurate initial guesses. Thanks to a new geometry based parameter, the resulting formulation is simplified and only requires one auxiliary function to handle the separate forms of the conic. Apart from enjoying 40% to 60% reduction in runtime over the current state-of-the art Gooding's method, the new formulation also

results in a robust and accurate implementation.

High-fidelity perturbation models are one of the major speed bottlenecks encountered during spacecraft trajectory design and simulation. The current work attempts to improve the performance of two aspects of these perturbation models, namely, the high-fidelity geopotential evaluation and the accurate ephemeris computation. High-fidelity geopotentials are typically computed via spherical harmonics, which is slow and non-intuitive to implement efficiently. In this thesis a new model called Fetch is proposed. Fetch is designed to take advantage of all the previous methods in the literature, while finding innovative solutions to correct their respective problems. The model is based on a modification to the Junkins weighting function method and achieves up to three orders of magnitude in speedup over the conventional spherical harmonics approach. As a part of this thesis, the Fetch model is applied to interpolate the GRACE GGM03C gravity model. Four Fetch models with different spherical harmonic degrees and order are computed and archived.

The next problem that deserves attention is the computation of accurate solar system body state and orientation data. The current work attempts to solve this problem by proposing a new ephemeris system called FIRE (Fast Interpolated Runtime Ephemeris). FIRE is custom designed for space trajectory applications that favor speed and smooth derivatives. It relies on spline interpolation and is based on a multi-level computation architecture. FIRE is demonstrated to be 50 to 70 times faster (compared to JPL's SPICE system) for typical trajectory applications while still achieving high accuracy. The speed is gained in exchange for a modest memory burden, which is necessary for the interpolation coefficients.

Shifting the focus to applications that require partial derivatives of a final state with respect to an initial state; the thesis also investigates the problem of fast sensitivity computation. Sensitivity information is used by many batch and sequential filtering applications, gradient based optimization algorithms, and is applied in a wide

range of engineering fields. The current work focuses on efficiently parallelizing sensitivity computation across a single trajectory. A new hybrid parallelization strategy is proposed, utilizing the Central Processing Unit (CPU) and the Graphics Processing Unit (GPU) to achieve rapid sensitivity computation on a single workstation. For example, trajectory propagation with overlapped computations demonstrate that first order sensitivities are calculated almost for free when compared to the conventional CPU implementation. The proposed technique can be applied to various optimization methods like optimal control, parameter optimization and other gradient based techniques.

The last chapter in this thesis aims to combine the two previously developed (Fetch and FIRE) perturbation models with a GPU based integration algorithm for simulating multiple high-fidelity space trajectories. The resulting tool provides unprecedented, multiplicative speedups over similar simulations on the CPU with performance gains of two to four orders in magnitude for various cases. The proposed tool is highly relevant to a variety of problems like space object conjunction analysis, covariance realism, particle filters and Monte-Carlo analyses.

CHAPTER I

INTRODUCTION

1.1 Motivation

Non-trivial, complex space trajectories[80,105,87,49,42] are being designed to tackle the increasingly challenging requirements and objectives of future space missions. A paradigm shift in the development of mission design algorithms is needed to handle future mission design and space catalog maintenance requirements.

The main thrust of this thesis is to participate in this paradigm shift by 1) identifying a manageable set of commonly used astrodynamics algorithms (referred to as building-block algorithms), which have a significant impact on the general space mission design process, 2) reformulating and developing novel solution methodologies to improve the accuracy and runtime performance of current state-of-the art algorithms and techniques and 3) taking advantage of modern computer hardware such as cluster computing and the Graphics Processing Unit (GPU) to deliver significant improvements to the current algorithms and techniques.

A total of five major problems are considered due to their wide applicability and potential for high impact in space-flight mechanics. The solutions to the first four problems act as building-block algorithms for solving larger and more complex problems in astrodynamics, while the fifth problem showcases an application for solving a large scale multiple spacecraft simulation problem. Improving and redesigning these building block algorithms can improve their associated software packages (both legacy and those yet to be written). The five selected problems along with their perceived approximate impact on various astrodynamics applications are listed in Table 1. Each of these problems have been extensively studied over the past few decades (details

are provided in the next section). Existing state-of-the art algorithms are typically classical in nature[48,101,13,37,11,101] and stand to benefit from the recent theoretical, algorithmic and computer hardware innovations. A dramatic improvement in performance can lead to cost savings for the whole mission design process and may help to solve the previously intractable problems in astrodynamics.

Table 1: Selected problems and their perceived potential impact (0 = no impact, 3 = high impact)

	Lambert's Problem	High-Fidelity Geopotential Computation	Ephemeris Computation	Sensitivity Computation	Multiple Spacecraft Trajectory Simulation
Preliminary mission design	3	1	2	2	1
Space surveillance	2	3	3	2	3
Trajectory optimization	3	2	2	3	2
Orbit determination	1	3	2	3	3
Tour design	3	0	2	1	1
Non-astrodynamic applications	0	2	2	3	0

In the next few sections a brief introduction to each of these five problems along with the relevant previous work and the adopted solution methodology is presented.

1.2 Previous Research and Proposed Solution Strategies

1.2.1 Lambert's problem

1.2.1.1 Introduction and literature review

In this thesis we start by tackling a basic, yet significant problem in space-flight mechanics, the multiple-revolution Lambert problem. The Lambert problem is one of the most extensively studied problems in celestial mechanics and astrodynamics.[78,48,13,139,37] Given two points that are relative to a point mass gravitating body and time of flight, a Lambert algorithm computes all possible Keplerian transfers between them. A variety of contributors have provided unique solution methods, yet the Lambert problem is always reduced to a one dimensional root-solve of a transcendental function. It was first introduced by Lambert in 1761 and was subsequently extended by Gauss.[139] With the arrival of the space age, the multiple revolution (multi-rev)

Lambert's problem (signifying multiple revolutions around the point mass) has since been studied and applied to a wide variety of space mission applications.

Due to its general formulation and wide applicability, the solution to the multi-rev Lambert problem acts as a building block for various problems like grand tour design,[138,84,116,130,75,53] interplanetary trajectory optimization,[1,84,115] and orbit determination.[86,14] Legacy codes that compute and optimize ballistic trajectories often require excessive numbers of Lambert solutions.[99,84,5] The inclusion of intermediate flybys and maneuvers further compounds the computational burden. The sheer number of Lambert calls needed can make the problem expensive to solve in terms of time and computer resources. Accordingly, to narrow the search space and provide for tractable problems, heuristic pruning techniques are generally required in practice. Optimal, and perhaps mission enabling solutions, therefore may be overlooked.

There exists a plethora of literature discussing various approaches developed over the years to solve the multi-rev Lambert problem. Most of these solutions techniques can be divided into two general types: 1) direct geometry based methods and 2) universal variable based methods. The types are characterized primarily by their choice of iteration variable. The direct geometry based methods iterate in the conventional space of orbit elements to solve some equivalent Lambert equations. Escobal[37] in his text gives multiple approaches where the iterates include semimajor axis, true anomaly, semiparameter, eccentricity and the f and g series. He only presents the zero-rev formulations but extending such techniques to multi-rev formulations is relatively straight forward. The work done by Ochoa and Prussing[93,32] during the early 1990s extends another geometry based approach, the Lagrange formulation, to its full multi-rev case. Their approach is robust and therefore it is commonly employed. However it suffers from three main drawbacks that are similar to most methods in its class: 1) the method is only valid for elliptical orbits, 2) the iteration variable is the semi-major axis and is therefore unbounded which can lead to numerical problems,

and 3) in the multi-rev case, the lower of the two solution branches is not single valued in flight time, causing notational complexity and a tedious implementation. A recent addition to this class of methods is the eccentricity-vector based solution by Avanzini[9] which has been more recently extended to multi-revs by He et al.[52] Other approaches in this category include the p-iteration method by Herrick and Liu[54] and the classical method by Gauss himself.[45] A dynamical systems based solution to the Lambert problem was also proposed by Nelson.[90] A recent solution strategy includes the series inversion solution by Thorne.[133]

The need for a universal (valid for all conics) and numerically robust solution technique led to the "universal variable" solution of the Lambert problem.[65,13,77,48,11,74] This approach is an efficient alternative solution method that combats many of the shortcomings of the Lagrange and other methods that iterate directly on orbital elements. The transformation to an auxiliary variable that is better behaved than one of the anomalies was introduced by Sundman.[131] This transformation was later applied to a "unified" time of flight formulation (for all types of conics) by Battin[13] and Bate.[11] It was shortly thereafter that Lancaster and his colleagues published in their short note[78] (and later in a more detailed technical report[77]) the first universal solution to the complete multi-rev Lambert problem. Battin was soon to follow with his own universal approach that was highly tuned for computer implementation using hypergeometric functions and continued fractions.[13,12] Meanwhile, Gooding[48] extended the work of Lancaster by formulating an initial guess generator and the higher order Halley's method for rapid root solving. Battin's method is mathematically elegant and computationally efficient, yet is not as intuitive as other approaches. Gooding was able to produce a robust, accurate Lambert algorithm which relied on first, second and third derivatives of the Lambert TOF function for rapid convergence. Another approach first proposed by Bate, Mueller and White[11] utilizes a simple transformation on the standard universal variable and results in just one root function that is valid

for all revolutions. The search variable is related to the transfer angle and is therefore easily bounded, except in the case of the hyperbola. Like all universal formulations, the full domain of the iteration variable is single valued. Overall, the implementation is straight-forward and computationally efficient. Recent studies comparing various Lambert formulations suggest that Gooding's implementation is the most robust and fastest implementation available. Specifically, a recent study done by Peterson et al.[98] evaluates the performance of 6 different methods (including Battin's and Gooding's method) for the zero revolution case. Their conclusion suggests that Gooding's method is the best (arguably if not outright) in terms of accuracy, robustness and speed. Similar conclusions were reached by Klummp[70] in a previous study. A recent implementation, documented in an article by DerAstrodynamics [1], claims to be "superior" to other Lambert algorithms but fails to provide any evidence in its support. Figure 1 briefly summarizes some of the major developments over the years related to solving the Lambert problem.
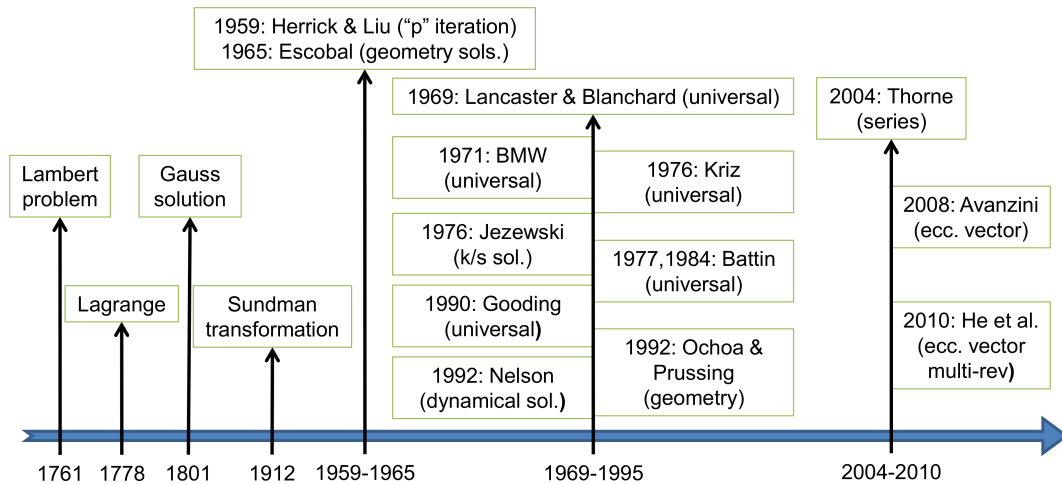
Figure 1: Timeline of major previous work related to solving the Lambert problem

[1]http://www.amostech.com/TechnicalPapers/2011/Poster/DER.pdf

1.2.1.2 *Overview of solution strategy and results*

With the intent to simplify the Lambert problem formulation and increase its runtime performance, a new formulation based on a cosine of the change eccentric anomaly is proposed. The new transformation simplifies the TOF equation and uses a bounded geometry parameter which dictates the shape of the TOF function. Accurate initial guess strategies based on rational functions lead to a fast and robust solution procedure. The proposed implementation is compared to Gooding's method, which is arguably the fastest and most robust.[70,98] The new formulation is derived in chapter 2 of this thesis, followed by a thorough performance analysis, comparing both accuracy and runtimes to the state of the art Gooding's method. Rapid root solve coupled with elegant derivative expressions results in 1.85, 1.75 and 2.15 times speedup (on average) over the Gooding method for hyperbolic, zero revolution and multiple revolution case, respectively.

1.2.2 High-fidelity geopotential computation

1.2.2.1 *Introduction and literature review*

A major hurdle towards fast high-fidelity trajectory simulation are the computationally slow spherical harmonics (SH) gravity fields computations.[101,27,85] The problem is expected to compound with the release of new, higher order gravity field models.[132,55] With the current computational resources it is typically not feasible to account for most of the accurate geopotential models in many high-fidelity trajectory simulations, such as monitoring the space catalog.[56] Accordingly, there is a pressing need for a new class of gravity models that can achieve orders of magnitude in speedup over SH while still preserving the accuracy and robustness of the SH approach.

Various alternatives to SH have been proposed and can be broadly divided into two classes 1) Discrete mass models and 2) Interpolation models. Discrete mass models (deemed as any model using point masses, or surface or volume mass distributions)

only require a limited amount of global data[71,144,107,106,88,145] and are straight forward to evaluate. They can be easily coupled with SH models for the benefit of an adaptive local resolution. Recently, there has been a growing interest in robotic and potential human exploration of small, irregular shaped bodies.[105] The volume-based and surface-based discrete mass models are therefore compelling because they are valid in the entire domain while SH representations fail to converge inside the reference radius.[140,96] In particular, the polyhedral method[96] is a robust and elegant solution for irregular small bodies although the extra computational requirements are cumbersome. Recently, a modern high-fidelity geopotential mascon model has been demonstrated to provide an order of magnitude speed improvements over SH when implemented on a common Graphics Processing Unit (GPU).[112]

The second and a more popular class of alternative potential formulations are the interpolation models. Applicable to both irregular and near-spherical shaped bodies, methods in this class expedite computations by effectively trading computer memory for runtime speed. Essentially first proposed by Junkins in 1976,[67] geopotential interpolation methods have been bolstered recently by the extraordinary memory resources of common computers. Depending on the method, a variety of techniques and basis functions have been employed including weighting functions,[69,68] wavelets,[19] splines,[19,81] octrees,[28] psuedocenters[59] and 3D digital modeling.[94] Each interpolation method balances accuracy with efforts to minimize runtime speed and memory footprint while achieving exactness, continuity and smoothness as appropriate.

The 3D geopotential interpolation model developed in this study is primarily motivated by the works of Junkins,[67,68] Hujsak,[59] Colombi et al.,,[28] Oltrogge[94] and Beylkin and Cramer[19] among others. Over thirty years ago Junkins demonstrated that a one order of magnitude speedup was possible at the expense of a modest investment in memory. This was a remarkable feat considering the quality of computers at the time. His model encompassed the region around Earth out to 1.2 radii and achieved roughly

6 digits of accuracy at the expense of storing 30,000 coefficients. Hujsak revisited the problem approximately 20 years later introducing the concept of interpolating the pseudocenter coordinates instead of the potential or acceleration. With a simpler interpolating scheme and improved hardware he fit a 70×70 field (for the northern hemisphere at altitudes between 400 and 1,500 km) with only 5 megabytes of storage. The algorithm requires the calculation overhead equivalent to a 5×5 field, leading to approximately a 100 fold speedup compared to SH for the 70×70 resolution. Columbi, Hirani, and Villac recently applied similar concepts using modern tools for calculating gravity fields for highly non-spherical bodies such as comets and asteroids. Their methods are demonstrated to provide approximately 100 fold speedups except their gains are compared to the expensive polyhedral methods. Beylkin and Cramer show recent progress in the efficient storage of multi-resolution interpolating functions and subsequently published the first global modern 3D interpolation geopotential model called the Cubed-Sphere model.[66] Using multiple concentric shells and Chebyshev basis functions, their method demonstrates 30 fold speedups compared to SH for their highest resolution 150×150 model. Their break even model in terms of runtime is said to be approximately at the resolution of 20×20. The memory footprint of their 150×150 model is 856 MB. Their model suffers from small discontinuities across shell boundaries, and requires the storage of extra coefficients for acceleration and higher order derivatives.

1.2.2.2 Overview of solution strategy and results

The main drawback of 3D geopotential interpolation gravity models is their memory requirements. As memory and processor technology has exponentially grown over the years, a renewed interest in memory-intense numerical methods is increasingly justified. Given the potential benefits of trading the abundantly available memory for tremendous speed improvements, a new hybrid model is proposed (called Fetch)

which attempts to take advantage of all the previous models, while finding innovative solutions to correct their respective problems. Accordingly, the Fetch model improves on the shortcomings of the aforementioned interpolation models. Priority is placed on a solution method that 1) is continuous and smooth across the global domain to an arbitrary order of derivatives, 2) is adaptive in terms of local vs. global resolution 3) has a residual error profile that is in the noise of the accuracy of the underlying base model (SH in this study), and 4) is singularity free. No previous or existing gravity field interpolation model can claim a complete handle on each of these ambitious priorities.

The new Fetch model optimally trades memory for speed and achieves global continuity by taking advantage of a weighted interpolation scheme (developed by Junkins et al.[69]). Singularity and associated numerical problems near the poles found in spherical coordinates is tackled using a two level overlapping global grid structure. High order analytic inverses for the resulting least squares problem are generated using an algebraic manipulator to ensure accurate and rapid coefficient evaluation. A parallel coefficient generation algorithm on a non-uniform grid (implemented using MPI) enables high-fidelity global geopotential Fetch model generation. The original Junkins weighting function method is modified to handle this new memory-saving, non-uniform grid. Chapter 3 goes over the Fetch interpolation approach and applies it to the geopotential application. Four Fetch models, based on efficient interpolation of the GRACE GGM03C SH gravity model[132] are generated. Up to three orders of magnitude in speedup over spherical harmonics is demonstrated with the memory requirements spanning from 120 MB to 2,360 MB.

1.2.3   Ephemeris computation

1.2.3.1   Introduction and literature review

Solar system ephemeris body states and orientations are consistently used in a variety of aerospace applications (for example[63, 46, 113, 114, 3, 83]). The importance of the

9

solar system data, the robust reliability of the current ephemeris systems (like JPL's SPICE[2]) and the need for precise trajectory computations have led to wide spread use of ephemerides amongst scientists, engineers, and their associated software. The current state-of-the-art ephemeris system in wide use today for high precision applications is the SPICE system provided by JPL. Similar SPICE related products from JPL such as the DE405 customized routines by Miles[128] are also commonly used. The SPICE system has not been designed for speed and fast ephemeris data retrieval but rather serves a much broader purpose. The SPICE system has a large collection of routines which can be used to read SPICE ephemeris files (for natural bodies and spacecraft). The system also provides information regarding the derived observation geometry such as altitude, latitude/longitude and the lighting angles of these bodies.

While SPICE and its derivative products are well-maintained and provide invaluable capabilities to users around the world, it is well-known that ephemeris calls are generally the bottleneck to speed improvements for precise applications. Typical trajectory simulation applications including optimization, differential correction, orbit determination and Monte-Carlo analysis often require thousands or more trajectory propagations using common force models and time spans. Considering each single iterate may easily require millions of calls to an ephemeris system, an extraordinary amount of computational resources is wasted on unnecessary overheads. While a broad spectrum of applications are indifferent to the computational burden of general ephemeris calls, many applications are simply bogged down. However, the extra cost has typically been considered an acceptable trade for the precision force model afforded by the accurate ephemerides.

The work presented in chapter 4 proposes a new custom ephemeris system that maintains the heavily relied upon accuracy, yet eliminates or substantially reduces the typical computational burdens associated with ephemeris calls. The new system, called FIRE (Fast Interpolated Runtime Ephemeris), is designed for custom trajectory applications that favor speed and smooth derivatives. The new system minimizes the overhead associated with ephemeris calls through the use of archived splines, a runtime ephemeris (stored in the random access memory of the computer) and batch processing routines. Performance comparisons with the Jet Propulsion Laboratory's Spacecraft Planet Instrument C-matrix Events (SPICE) ephemeris system show typical speed improvements of up to approximately two orders of magnitude. Performance comparisons for high-fidelity trajectory propagations are also considered and a factor of 70 in performance increase is achieved for typical cases. FIRE has potential value to any high precision application or software requiring fast, accurate and smooth ephemeris data.

1.2.4 Sensitivity computation

1.2.4.1 Introduction and literature review

Mathematical and computational models are used in most areas of science and engineering for performing optimization.[117,142,40,119] Gradient based numerical optimization relies on accurate sensitivity information to robustly move a solution towards an optimum. While there are various subfields in numerical optimization such as Optimal Control[22,110,109,123,104] and Parameter Optimization,[60,47] all gradient based continuous methods make use of numerical sensitivities to select new step directions.

Specifically, many trajectory optimization algorithms rely heavily on higher order sensitivity information.[100,18,39,20] The computational burden of sensitivity calculations increases exponentially with problem complexity and the requirement for

higher order derivatives. Therefore, with the existing CPU architecture, it is often not feasible to solve realistic model problems because of the extraordinarily expensive sensitivity calculations. Given a function evaluation computational complexity of $O(n)$, the corresponding first order sensitivities have a computational complexity of $O(n^2)$, and similarly second order sensitivities have a computational complexity of $O(n^3)$. This complexity (and the high costs of large CPU clusters required) therefore prohibits many classes of high-fidelity optimization problems from being solved.

Parallel sensitivity analysis has been limited to a narrow class of problems.[21, 24, 20, 29] These methods are either not scalable or they perform inefficiently as the current CPU hardware is not able to exploit the massive parallelism present in the underlying problem. To further improve performance, a reformulation of the solution method along with exploitation of the current state of the art computer hardware is required.

1.2.4.2 Overview of solution strategy and results

In this thesis a novel strategy to use NVIDIA's GPU (Graphics Processing Unit) to rapidly calculate the sensitivities (derivatives) in a multilayer, parallel, and heterogeneous way while the CPU (Central Processing Unit) sequentially computes the less expensive state equations, is proposed. The proposed tool computes both the first and second order analytic sensitivities on the GPU with double precision accuracy. The results show that first order sensitivities are calculated almost for free and an order of magnitude speedup is obtained for second order sensitivities when compared to a conventional CPU implementation. More details on the novel overlapping solution strategy, its implementation and associated performance are given in chapter 5.

1.2.5 Multiple Spacecraft Trajectory Simulation

1.2.5.1 Introduction and literature review

With the advent of modern computers the field of computational space flight mechanics has witnessed rapid growth.[33, 8, 4, 66, 112, 92] This growth has fueled the need

for high-fidelity spacecraft trajectory design and simulation tools, which are gaining wider acceptance in the astrodynamics research community. These tools and methodologies are becoming more relevant as the space environment is getting more crowded, and as the surveillance sensors are improving in accuracy. Current state-of-the-art high-fidelity integration algorithms and force models are computationally burdensome, and often require large CPU computing clusters. High fidelity models are therefore under-utilized in a variety of space surveillance applications, such as real time tracking, the conjunction problem, particle filters, orbital debris tracking, etc. A paradigm shift capable of providing multiple orders of magnitude in speedup while still maintaining the desired accuracy, would be a significant step forward in solving these computationally challenging problems.

The speed bottlenecks in most of the mentioned applications can be divided into two main components: 1) trajectory integration and 2) force model computation. Both of these sub problems have been studied by various authors over the years. See for example, the references.[61,10,44] There exist a large number of numerical integration techniques in the literature and they can be broadly divided in to two types 1) single-step methods[36,50] and 2) multi-step methods.[17,102] Compared to single-step integrators, the multi-step integrators are more complicated to implement, but have the advantage of being more efficient as they can take larger step sizes for a given accuracy, and also require fewer evaluations per step.[17] One of the most commonly used single-step methods for performing high-fidelity integration is the high order Runge Kutta technique.[36,73] Over the years various other alternatives, like the collocation methods[10] or the Taylor series integration methods,[120] have also been proposed. These methods show promise but further research is needed prior to a broad acceptance by the community. The problem of fast force model computation has also been subjected to numerous studies in the past.[71,112,88,68,8,6,19] Often truncated[16,26] and semi-analytic techniques[103,61] are adopted, which compromise the accuracy of higher

order solutions to improve performance.

1.2.5.2 Overview of solution strategy and results

The aim of chapter 6 of this thesis is to bring together the fast ephemeris computation model (chapter 4) and the new Fetch gravity model (chapter 3), along with novel parallel algorithms to develop an integrated tool that is capable of performing high-fidelity integration of multiple spacecraft at unprecedented speeds. To achieve this goal a spacecraft integration tool is proposed that takes advantage of 1) new fast and accurate gravity perturbation models (the FIRE and Fetch model) and 2) a Graphics Processing Unit (GPU) based Runge-Kutta integrator to achieve massive parallelism across multiple spacecraft. The two methods combined lead to multiplicative speedups, making the tool two to four orders in magnitude faster, when compared to a similar simulation performed on a single CPU. The solution methodology is highly relevant to the conjunction problem, covariance realism, particle filters and Monte-Carlo analyses.

1.3   Dissertation Organization

This dissertation is divided into 5 primary chapters (2 to 6), excluding the introduction and the conclusion. Each chapter is dedicated to solving one of the five problems listed in Table 1. Each of these problems presented challenges which were both theoretical and computational in nature. Keeping this in mind, a major part of the current work has been to find solution methodologies which provide orders of magnitude in speed increase while maintaining accuracy and robustness. Apart from novel solution strategies, modern computational techniques like volume interpolation, analytic matrix inversions and parallel GPU and CPU based programming have also been utilized. Chapter organization, the type of improvements and the computer resources used are summarized in Table 2.

Table 2: Chapter organization and classification of contributions

Problem	Chapter	Contribution type	Hardware used
Lambert's problem	2	theoretical, algorithmic	CPU
High-fidelity geopotential computation	3	theoretical, methodological, algorithmic	CPU, Cluster computing
Ephemeris computation	4	methodological, algorithmic	CPU
Sensitivity computation	5	methodological,algorithmic	CPU and GPU
Multiple spacecraft trajectory simulation	6	algorithmic, methodological	CPU and GPU

1.4   Publication History

The work done in this dissertation has been presented at various technical conferences. Work done in chapter 4 has already been published in the journal of Celestial Mechanics and Dynamical Astronomy. A publication based on chapter 3 has been submitted to an AIAA journal and is currently under review. Appendix C lists the author's publication history.

CHAPTER II

THE MULTIPLE REVOLUTION LAMBERT PROBLEM

2.1 *Chapter Summary*

In this chapter, a new universal variable (k) is introduced to improve the solution performance for the multiple revolution Lambert boundary value problem. The formulation is motivated by the Bate, Muller and White's universal variable approach and is based on the cosine of the change in eccentric anomaly. The formulation takes advantage of a geometry-based parameter to simplify the universal formulation of the Lambert equation. This equation, defined as a function of the universal variable k, is shown to have simplified derivative expressions and requires only a single transcendental function evaluation. The two Stumpff functions, present in several classic formulations, are condensed to a single analogous function, that handles the separate conic. Using rational polynomials for initial guesses combined with Halley's method for root-solving, allows for convergence in 2-4 iterations for $\sim$99.5% of cases (on average). The accurate initial guess approximations further reduce the number of minimization calls that are typically needed in order to bound the multiple revolution root-solve. The formulation is derived, followed by a thorough performance analysis, comparing both accuracy and runtimes to the state of the art Gooding's method. The method proposed is demonstrated to be statistically as accurate as the Gooding method, while achieving $\sim$40-60% reductions in runtime, on average.

2.2 *Chapter Nomenclature*

x	Sundman transformation variable
k	Proposed universal variable
a	Semimajor axis
e	Eccentricity

p	Semilatus rectum
θ	Transfer angle
d	Transfer angle parameter ($+1$ for $0 < \theta < \pi$, -1 for $\pi < \theta < 2\pi$)
f, g	Lagrange f and g functions
τ	Lambert geometry parameter
ΔE	Change in eccentric anomaly
ΔF	Change in hyperbolic anomaly
i	i^{th} spacecraft revolution
M	Mass of the body
N	Number of spacecraft revolutions
GM	Gravitational parameter of the primary
T^*	Target time of flight
k^*	Value of k corresponding to T^*
$\tilde{k}^*$	Initial guess for k^*
k_i^*	Value of k corresponding to T^* for the ith revolution transfer
$\tilde{k}_i^*$	Initial guess for k_i^*
TOF	Time of flight
TOF'	First derivative of TOF function with respect to k
TOF''	Second derivative of TOF function with respect to k
$T_{b,i}$	Minimum TOF for the ith revolution transfer
$k_{b,i}$	k corresponding to $T_{b,i}$
$\tilde{k}_{b,i}$	Approximation for $k_{b,i}$
$\vec{r_1}$	Initial position vector
$\vec{r_2}$	Final position vector
$\vec{v_1}$	Initial velocity vector
$\vec{v_2}$	Final velocity vector
W	Auxiliary function
W'	First derivative of W with respect to k
W''	Second derivative of W with respect to k
N_{max}	User requested maximum number of revolutions
N_{ub}	Total number of revolutions possible for a given set of input variables
zero-rev	Zero revolution
lp	Long period
sp	Short period
multi-rev	Multiple revolution
wrt	With Respect To

2.3   Introduction and Background

Lambert's problem enjoys the title of being one of the extensively studied problems in celestial mechanics and astrodynamics.[78,48,13,139,37]   The problem has wide applicability and its solution acts as a building-block algorithm for various problems like interplanetary trajectory design and optimization,[1,84,115] grand tour design,[138,84,109,130,75,53] and orbit determination[86,14]

The general Lambert problem involves computing all possible keplerian transfers between two points relative to a point mass gravitating body. The general solution procedure involves a one dimensional root-solve of a transcendental equation, which is commonly addressed as the Lambert time of flight equation. The solutions methods existing in literature for solving the multiple revolution Lambert problem fall under two general classes 1) direct geometry based methods and 2) universal variable based methods. Direct geometry based methods iterate on one of the conventional orbital elements and have been a subject of active research for many years.[37,45,93,32,54,9,52] The need for a universal formulation, valid for all conics lead to the development of the universal variable based solution of the multi-rev Lambert problem. The universal variable based solution method is numerical robust and has enjoyed many outstanding contributions over the past few decades.[65,13,77,48,11,74] Recently, an alternate class of methods based on dynamical systems theory[90] and series inversion[133] have also been proposed.

Recent comparative studies performed by Peterson et al.[98] and Klumpp both identify Gooding's method to be one of the most fastest and accurate implementations available for the solving the Lambert problem for the zero revolution case. Gooding's[48] method extends the work of Lancaster[77] by formulating an initial guess generator and uses a 2nd order method for rapid root solving.

In this chapter a formulation of the multiple revolution Lambert problem based on a cosine of the change eccentric anomaly, is introduced. The formulation is motivated

by Bate et al's.[11] universal variable approach and shown to simplify the Lambert TOF equation. Further, it also identifies a new bounded geometry parameter which dictates the shape of the TOF function. The proposed formulation requires only a single transcendental function evaluation and enjoys simple, elegant derivative expressions. Zero-rev and multi-rev initial guess strategies based on rational functions lead to a fast and robust solution procedure. The accurate multi-rev initial guess on a geometry based parameter is further used to significantly reduce the number of minimization calls that are typically needed in order to bound the multiple revolution root-solve. Given the simplified form of the TOF equation and its higher order derivatives, a second order method (similar to the Gooding's method) is used for rapid root solving of the TOF equation. The proposed implementation is compared to Gooding's method, which as stated before is arguably the fastest and most robust. Furthermore, the comparisons performed in this chapter use the original code published by Gooding[48] leading to a fair comparison. Thanks to the accurate initial guesses, the proposed method (like Gooding's method) also converges in 3 iterations for a vast majority of the cases. Rapid convergence coupled with a simplified formulation results in 1.85, 1.75 and 2.15 times speedup (on average) over the Gooding method for hyperbolic, zero-rev and multi-rev case, respectively.

In the next section a briefly derivation of the Lambert formulation is presented. Next, the initial guess generation techniques are described, followed by algorithm implementation details. Finally, the performance of the new formulation is presented and compared with Gooding's method on a statistically exhaustive set of transfers.

2.4   Multiple Revolution Lambert Problem

Figure 2 depicts a general diagram of the Lambert problem. Vectors $\vec{r_1}$ and $\vec{r_2}$ are the bounding position vectors; TOF stands for the time of flight for the transfer and θ is the transfer angle between the two position vectors. All possible arcs connecting the
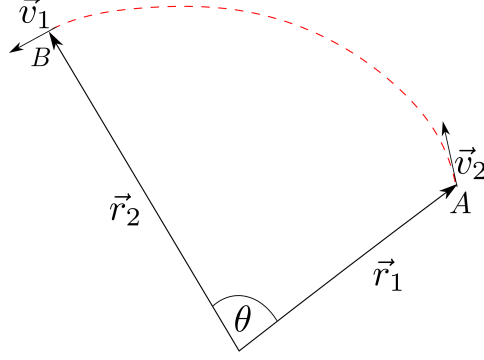
$\vec{v}_1$ and $\vec{v}_2$ are obtained after solving the Lambert problem

Figure 2: General problem geometry

two points A and B with the required TOF and which satisfy Keplerian motion are possible Lambert solutions. There are $2N_{ub}^D+1$ direct solutions and $2N_{ub}^R+1$ retrograde solutions where N_{ub}^D and N_{ub}^R is the maximum number of revolutions possible for direct and retrograde transfers, respectively, given a set of input parameters.

2.4.1   The Lambert formulation

In this section, a Lambert formulation based on the new universal variable, "k", is derived. The formulation is motivated by Bate et al's.[11] universal variable approach and follows a similar derivation procedure to obtain the Lambert time of flight equation.

As Keplerian motion is confined in a plane, the vectors $\vec{r_2}$ and $\vec{v_2}$ can be expressed as a function of $\vec{r_1}$ and $\vec{v_1}$, given by Eqs. 1 and 2.

$$\vec{r_2} = f\vec{r_1} + g\vec{v_1} \tag{1}$$

$$\vec{v_2} = \dot{f}\vec{r_1} + \dot{g}\vec{v_1} \tag{2}$$

where f, g, are the Lagrange f and g functions, respectively. The f and g functions assume vectors $\vec{r_1}$ and $\vec{v_1}$ form the basis of the resulting solution space. The f and g expressions can be written as a function of both the transfer angle θ and the change

in eccentric anomaly ΔE (see pgs. 218-219 in Bate et al.[11]) , given by Eq. 3-6 below:

$$f = 1 - \frac{a}{r_1}[1 - \cos(\Delta E)] = 1 - \frac{r_2}{p}[1 - \cos(\theta)] \qquad (3)$$

$$g = TOF - \sqrt{\frac{a^3}{\mu}}[\Delta E - \sin(\Delta E)] = \frac{r_1 r_2 \sin(\theta)}{\sqrt{\mu p}} \qquad (4)$$

$$\dot{f} = -\frac{\sqrt{\mu a}\sin(\Delta E)}{r_1 r_2} = \sqrt{\frac{\mu}{p}}\frac{1 - \cos(\theta)}{\sin(\theta)}\left[\frac{1 - \cos(\theta)}{p} - \frac{r_1 + r_2}{r_1 r_2}\right] \qquad (5)$$

$$\dot{g} = 1 - \frac{a}{r_2}[1 - \cos(\Delta E)] = 1 - \frac{r_1}{p}[1 - \cos(\theta)] \qquad (6)$$

The classic Sundman transformation defines the time rate of change of the universal variable x (Eq. 7) and is expressed as a function of change in eccentric anomaly ΔE (see pgs. 191-192 in Bate et al.[11]), given in Eq. 8:

$$\dot{x} = \frac{\sqrt{\mu}}{r} \qquad (7)$$

$$\frac{x}{\sqrt{a}} = \Delta E \qquad (8)$$

As the hyperbolic anomaly (F) and the eccentric anomaly (E) are related via the definition $F = iE$, the equivalent form of Eq. 8 for hyperbolic orbits is given by:

$$\frac{x}{\sqrt{-a}} = \Delta F \qquad (9)$$

We now introduce the new universal variable k, defined by the transformation $\cos(\Delta E) = k^2 - 1$, given by the right triangle in Fig. 3
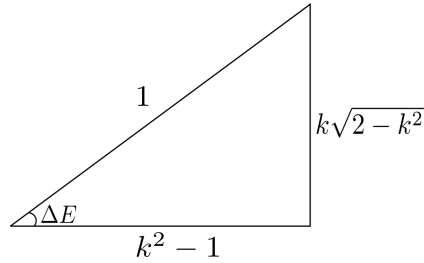
Figure 3: Pythagorean transformation triangle

Using Eq. 8 and 9 we relate the universal variable k with x as follows:

$$\cos(\frac{x}{\sqrt{a}}) = \cos(\Delta E) = k^2 - 1 \tag{10}$$

$$\sin(\frac{x}{\sqrt{a}}) = \sin(\Delta E) = k\sqrt{2 - k^2} \tag{11}$$

$$\cos(i\frac{x}{\sqrt{-a}}) = \cosh(\Delta F) = k^2 - 1 \tag{12}$$

$$\sin(i\frac{x}{\sqrt{-a}}) = \sinh(\Delta F) = k\sqrt{k^2 - 2} \tag{13}$$

For ease of derivation, an intermediate variable, q, is defined as follows:

$$q = \frac{x}{\sqrt{a}} = \Delta E = -i\Delta F \tag{14}$$

Taking into account the correct quadrant and using Eqs. 10, 12 and 14, q can be expressed as a function of k and N as follows:

$$q = \begin{cases} (1 - \text{sgn}(k))\pi + \text{sgn}(k)\cos^{-1}(k^2 - 1) + 2\pi N & -\sqrt{2} \le k \le \sqrt{2} \\ -i\cosh^{-1}(k^2 - 1) & k \ge \sqrt{2} \end{cases} \tag{15}$$

Note that the variable q is also related to the universal variable, z, used by Bate et al.,[11] as $z = q^2$. Using Eq. 14, the expression for a can be written as:

$$a = \frac{x^2}{q^2} \tag{16}$$

Substituting a from Eq. 16 into Eq. 3 results in the following expression:

$$\frac{x^2}{q^2 r_1}[1 - \cos(\Delta E)] = \frac{r_2}{p}[1 - \cos(\theta)] \tag{17}$$

Using the definition of $\cos(\Delta E)$ from Eq. 10 and isolating the expression for p from Eq. 17, we obtain:

$$p = \frac{r_1 r_2 q^2 (1 - \cos(\theta))}{x^2 (2 - k^2)} \tag{18}$$

Substituting p from Eq. 18 into Eq: 5, the following expression is obtained:

$$-\frac{\sqrt{\mu a}\sin(\Delta E)}{r_1 r_2} = \sqrt{\frac{\mu x^2}{r_1 r_2 q^2}} \frac{\sqrt{[1 - \cos(\theta)](2 - k^2)}}{r_1 r_2 \sin(\theta)} \left[\frac{x^2 (2 - k^2)}{q^2} - (r_1 + r_2) \right] \tag{19}$$

22

Next, substitute $\sin(\Delta E)$ from Eq. 11 and a from Eq. 16 into Eq. 19 and simplify to obtain:

$$-k\left(\frac{\sqrt{r_1 r_2}\sin(\theta)}{(r_1 + r_2)\sqrt{1 - \cos(\theta)}}\right) = \frac{x^2(2 - k^2)}{(r_1 + r_2)q^2} - 1 \tag{20}$$

Inspecting the LHS of Eq. 20 we see the possibility of zero both in the numerator and the denominator when $\theta = 0$. To circumvent this, following trigonometric equivalence is implemented:

$$\frac{\sin(\theta)}{\sqrt{1 - \cos(\theta)}} = d\sqrt{1 + \cos(\theta)} \quad , \quad d = \begin{cases} +1 & 0 \leq \theta \leq \pi \\ -1 & \pi \leq \theta \leq 2\pi \end{cases} \tag{21}$$

Next, a parameter dependent only on geometry is introduced:

$$\tau = d\frac{\sqrt{r_1 r_2[1 + \cos(\theta)]}}{r_1 + r_2} \tag{22}$$

where τ is a non-dimensional function of r_1, r_2 and θ and varies between $\left[\frac{-1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$. Note, that the parameter τ is related to the unbounded parameter A found in the universal variable approach of Bate et al.,[11] as follows

$$\tau = \frac{A}{r_1 + r_2} \tag{23}$$

Using the above definition of τ, Eq. 20 simplifies into the following expression for x:

$$x = q\sqrt{\frac{(r_1 + r_2)(1 - k\tau)}{(2 - k^2)}} \tag{24}$$

Substituting a and p from Eqs. 16 and 18, respectively, into Eq. 4, leads to:

$$TOF = \frac{(r_1 + r_2)}{\sqrt{\mu}}\frac{x}{q}\sqrt{2 - k^2}\left[d\frac{\sqrt{r_1 r_2[1 + \cos(\theta)]}}{r_1 + r_2} + \frac{x^2(q - k\sqrt{2 - k^2})}{q^2\sqrt{2 - k^2}(r_1 + r_2)}\right] \tag{25}$$

Substituting τ from Eq. 22 and x from Eq. 24 into Eq. 25 results in the following expression for TOF as a function of input geometry variables (τ and S) and the

universal iterate variable, k:

$$TOF(k) = S\sqrt{1 - k\tau}\left[\tau + (1 - k\tau)W\right] \tag{26}$$

$$S = \sqrt{\frac{(r_1 + r_2)^3}{\mu}}$$

$$m = 2 - k^2$$

$$W = \frac{q}{\sqrt{m^3}} - \frac{k}{m}$$

Note that W becomes indeterminate as k approaches $\sqrt{2}$ and thus requires a series evaluation when $k - \sqrt{2}$ is close to zero. Using q from Eq. 15, W is expressed as follows:

$$W = \begin{cases} \frac{(1-\mathrm{sgn}(k))\pi + \mathrm{sgn}(k)\cos^{-1}(1-m) + 2\pi N}{\sqrt{m^3}} - \frac{k}{m} & -\sqrt{2} \le k < \sqrt{2} - \epsilon \text{ (elliptical orbits)} \\ \frac{-\cosh^{-1}(1-m)}{\sqrt{-m^3}} - \frac{k}{m} & k > \sqrt{2} + \epsilon \text{ (hyperbolic orbits)} \\ W_s & \sqrt{2} - \epsilon \le k \le \sqrt{2} + \epsilon \ (N = 0) \end{cases} \tag{27}$$

where W_s is given by the following series:

$$W_s = \frac{\sqrt{2}}{3} - \frac{v}{5} + \frac{2}{35}\sqrt{2}v^2 - \frac{2}{63}v^3 + \frac{2}{231}\sqrt{2}v^4 - \frac{2}{429}v^5 + \frac{8}{6435}\sqrt{2}v^6 - \frac{8}{12155}v^7$$

$$+ \frac{8}{46189}\sqrt{2}v^8 - \frac{8}{88179}v^9 + \frac{16}{676039}\sqrt{2}v^{10} - \frac{16}{1300075}v^{11} + \frac{16}{5014575}\sqrt{2}v^{12} - \frac{16}{9694845}v^{13}$$

$$+ \frac{128}{300540195}\sqrt{2}v^{14} - \frac{128}{583401555}v^{15} + \frac{128}{2268783825}\sqrt{2}v^{16} + O\left(v^{\frac{33}{2}}\right) \tag{28}$$

$$v = k - \sqrt{2}$$

For the current thesis, the W_s series is truncated to include up to $O\left(v^8\right)$, and an ϵ value of 2E-2 is selected resulting in W_s with a maximum error of 2E-15 at the $\sqrt{2} - \epsilon$ boundary. It should be noted that W is related to the classic Stumpff functions (C and S), defined in Bate et al.,[11] by the following expression:

$$W = \frac{S}{C\sqrt{C}} \tag{29}$$

Equation 26 (referred to as the TOF equation) along with Eq. 27 complete the set of equations required to fully solve the multi-rev Lambert problem. The expression for TOF is an explicit function of the universal variable k and requires only one inverse

cosine function and 3 square roots to calculate the time of flight. Furthermore, S has the dimension of time, hence the expression to the right of S is dimensionless. This decoupling helps in the generation of initial guesses for the multi-rev transfer solution, as discussed later. Figure 4 shows a representative plot of the whole solution
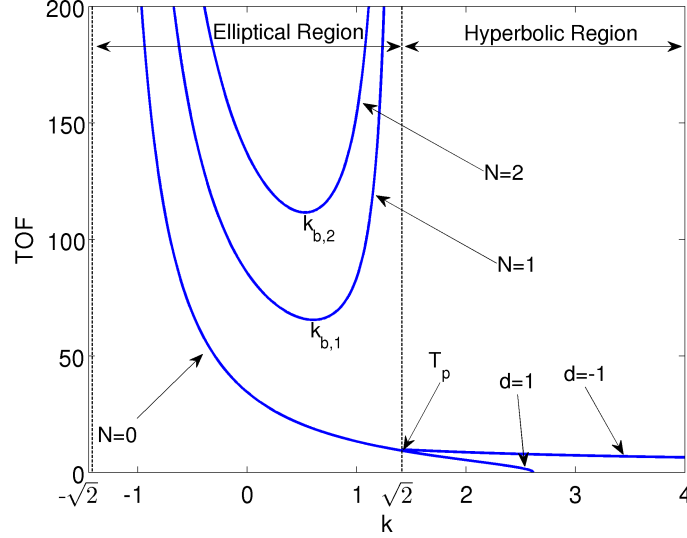
Figure 4: TOF vs. k $[r_1 = 1, r_2 = 7.098, \theta = 69.37 \ (\text{deg})]$

space as a function of k. Only the $d = 1$ case is shown for the elliptic transfers, while the hyperbolic cases include both $d = \pm 1$. In Fig. 4, T_p stands for the parabolic TOF and is obtained by substituting $k = \sqrt{2}$ in the TOF equation:

$$T_p = S\sqrt{1 - \sqrt{2}\tau} \frac{(\tau + \sqrt{2})}{3} \tag{30}$$

It should be noted that for $d = 1$, TOF goes to zero at $k = \frac{1}{\tau}$ but for $d = -1$, TOF goes to zero at $k = +\infty$. Furthermore, each multi-rev transfer has a minimum possible time of flight associated with it (T_{bi}), as shown in Fig. 4. Hence, if T^* is less than $T_{b,i}$ for a given revolution i, then the ith revolution transfer does not exist. Hence, for a given set of input parameters there exists a unique upper bound on N, defined as N_{ub}. If the solution exists, then the accurate location of the associated $k_{b,i}$ provides robust bounds for ensuring a successful root search on each branch.

2.4.2 Solution Procedure

Given a target time of flight (T^*) and an input geometry set $(S$ and $\tau)$; the solution to the Lambert problem is found via a one-dimensional root-solve of the function L, given below:

$$L(k) = TOF(k) - T^* \tag{31}$$

where k is the universal iteration variable defined in the previous section. All gradient based root solving methods require at least the first derivative of the function being solved. The multi-rev case, as shown in Fig. 4, has a minimum TOF value $(T_{b,i})$ for $i > 0$, and is obtained by minimizing L for each revolution i. These minimizations are achieved using Newton's method (requiring only the first two derivatives) and are necessary to identify N_{ub} and robustly bound the root solves of L in the multi-rev case. For a given value of k, f and g functions are used to calculate the final v_1 and v_2 velocity vectors as follows:

$$f = 1 - \frac{1 - k\tau}{r_1} \tag{32}$$

$$g = \tau(r_1 + r_2)\sqrt{1 - k\tau} \tag{33}$$

$$\dot{g} = 1 - \frac{1 - k\tau}{r_2} \tag{34}$$

$$\vec{v_1} = \frac{\vec{r_2} - f\vec{r_1}}{g} \tag{35}$$

$$\vec{v_2} = \frac{\dot{g}\vec{r_2} - \vec{r_1}}{g} \tag{36}$$

The decision to use a second order root solving method is justified due to the relatively simple form of the TOF function and its derivatives wrt k. Equation 37 gives these compact expressions (see Eq. 26):

$$TOF' = \frac{-TOF}{2c} + S\tau\sqrt{c\tau}(W'c - W)$$
$$TOF'' = \frac{-TOF}{4c^2} + S\tau\sqrt{c\tau}(\frac{W}{c} + cW'' - 3W') \tag{37}$$
$$c = \frac{1 - k\tau}{\tau}$$

where W' and W'' are the first and second derivatives of W wrt k, efficiently expressed in Eqs. 38 and 39, respectively.

$$
W' = \begin{cases} \frac{-2+3Wk}{m} & k < \sqrt{2} - \epsilon \\ \frac{-2+3Wk}{-m} & k > \sqrt{2} + \epsilon \\ \frac{\partial W_s}{\partial k} & \sqrt{2} - \epsilon < k < \sqrt{2} + \epsilon \end{cases} \tag{38}
$$

$$
W'' = \begin{cases} \frac{5W'k+3W}{m} & k < \sqrt{2} - \epsilon \\ \frac{5W'k+3W}{-m} & k > \sqrt{2} + \epsilon \\ \frac{\partial^2 W_s}{\partial k^2} & \sqrt{2} - \epsilon < k < \sqrt{2} + \epsilon \end{cases} \tag{39}
$$

The Halley's method enjoys cubic convergence unless $TOF'' \approx 0$, in which case it behaves like the Newton method. The iteration formula for the Halley's method is briefly stated is Eq. 40.

$$
k^{i+1} = k^i + \Delta k
$$

$$
\Delta k = -L \left[TOF' - \frac{LTOF''}{2TOF'} \right]^{-1} \tag{40}
$$

Note that Gooding[48] also uses Halley's method to achieve rapid convergence. In the next section the initial guess strategies for the hyperbolic case, zero revolution elliptical case, and multi-rev case, respectively.

2.5 Initial Guess Generation

A robust Lambert solver relies on a fast and efficient root-solving algorithm. However the convergence of any root solver benefits from a close initial guess. The lack of such starting points has led researchers to adopt other less efficient root solving algorithms (like the bisection method) for solving the Lambert's problem.[139] On the other hand, Gooding, using an accurate initial guess strategy is able to achieve convergence to at least 13 digits in 3 iterations for a vast majority of the cases.

Similar to Gooding's method strategy of initial guess generation,[5,48] the solution space is divided into various regions and accurate approximations of the TOF function

within those regions are generated.

We start by defining a general rational approximating function given by Eq. 41.

$$F(x) = \frac{ax^\alpha + 1}{bx^\alpha + c} \tag{41}$$

where x is normalized from 0 to 1 and $F(x)$ is the function being approximated. A simple linear transformation in Eq. 42 is used to convert from a normalized x to k. Variables k_n and k_m denote the bounds on the value of k for a given region.

$$k = k_n + (k_m - k_n)x \tag{42}$$

The coefficients a, b, and c are found enforcing general boundary conditions and intermediate constraints, $F_1 = F(1), F_i = F(x_i), F_0 = F(0)$ resulting in:

$$F(x) = \frac{[F_i(-F_1 + F_0)x^\alpha + F_0(F_1 - F_i)]x_i{}^\alpha - x^\alpha F_1(F_0 - F_i)}{[(-F_1 + F_0)x^\alpha + F_1 - F_i]x_i{}^\alpha - x^\alpha(F_0 - F_i)} \tag{43}$$

where x_i stands for some intermediate value of x between 0 and 1 and changes based on the region being approximated. Enforcing an intermediate constraint allows us to capture the changes in curvature of the function being approximated. Similar to x_i, α also changes based on the region being approximated and is selected empirically so as to allow for computation of the power $\frac{1}{\alpha}$ without the use of the expensive *pow* function. Having the same degree (α) in the numerator and denominator allows us to invert the rational polynomial (Eq. 41):

$$x^* = \left[\frac{Z(F_0 - F^*)(F_1 - F_i)}{(F_i - F^*)(F_1 - F_0)Z + (F_0 - F_i)(F_1 - F^*)} \right]^{\frac{1}{\alpha}} \tag{44}$$

$$Z = x_i^\alpha$$

It should be noted that for a given value of α and x_i, Z can be precomputed and stored as a constant. Finally, Eq. 42 is applied to convert from x^* to $\tilde{k}^*$.

$$\tilde{k}^* = k_n + (k_m - k_n)x^* \tag{45}$$

where $\tilde{k}^*$ is an approximation of k^* corresponding to T^*. For speed consideration all the initial guesses maybe computed in single precision.

Table 3 lists various Lambert problem sets considered for evaluating the performance of the initial guesses defined in the next 3 sections. Throughout this study length units (LU) and time units (TU) are selected such that r_1 and GM are normalized to unity. The normalized convergence tolerance is set to 1E-13, unless specified otherwise. The vector $\vec{r_1}$ is expressed as function of the three variables a, b and c (see Table 3):

$$\vec{r_1} = \left[\frac{a}{\sqrt{a^2+b^2+c^2}}, \frac{b}{\sqrt{a^2+b^2+c^2}}, \frac{c}{\sqrt{a^2+b^2+c^2}} \right] \tag{46}$$

Table 3: Test runs

Test Name	Sol. Type	# Test cases	a	b	c	$\vec{r_2}(1:3)$ (LU)	TOF (TU)	N_{max}
A	Hyperbolic	1,000,000	-10↔10	-10↔10	-10↔10	-10↔10	0.3↔35.25	0
B	Elliptical	1,000,000	-10↔10	-10↔10	-10↔10	-10↔10	0.0↔500.0	0
C	Elliptical	1,000,000	-9↔9	-9↔9	-9↔9	-9↔9	0.0↔1000.0	20
D	Elliptical	1,000,000	-9↔9	-9↔9	-9↔9	-9↔9	0.0↔2000.0	20
E	Elliptical, Hyperbolic	1,000,000	-5↔5	-5↔5	-5↔5	-10↔10	2.00↔1000.00	20

2.5.1 Hyperbolic initial guess generation
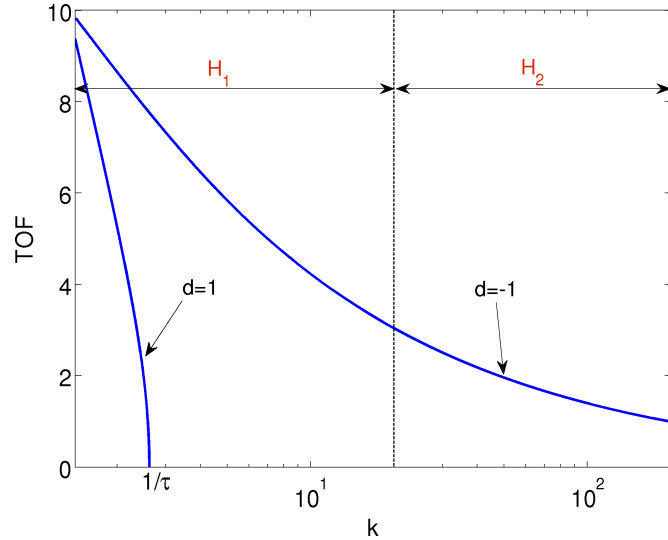
Figure 5: Hyperbolic solution space $[r_1 = 1, r_2 = 7.098, \theta = 69.37, 290.63$ (deg)$]$

Figure 5 gives a general overview of the hyperbolic solution space as a function of k. The general behaviour of the solution space changes with the parameter, d, and

hence separate initial guesses are generated for $d = \pm 1$, respectively. For $d = 1$, TOF goes to zero as k approaches $\frac{1}{\tau}$, leading to well defined search boundaries. For $d = -1$, the solution space is divided into two regions (see Fig. 5) as follows:

- Region H_1 : $\sqrt{(2)} \le k \le 20$

- Region H_2 : $20 < k \le +\infty$

For $d=1$ and $d=-1$ (only region H_1) , we use the general initial guess formula given by Eq. 44.

<div align="center">

Table 4: Hyperbolic initial guess parameters

Case	Region	Eq.#	k_n	k_m	k_i	Z	α	F_0	F_1	F_i	F^*
$d = 1$	H_1, H_2	44	$\sqrt{(2)}$	$\frac{1}{\tau}$	$\frac{k_n+k_m}{2}$	$\frac{1}{\sqrt{2}}$	$\frac{1}{2}$	T_p (see Eq. 30)	0.0	$TOF(k_i)$	T^*
$d = -1$	H_1	44	$\sqrt{(2)}$	20	$\frac{2k_n+k_m}{3}$	$\frac{1}{3}$	1	T_p	$TOF(20)$	$TOF(k_i)$	T^*
$d = -1$	H_2	47	-	-	-	-	-	-	-	-	-

</div>

For the case $d = -1$ and region H_2, a modified form of Eq. 44 (given by Eq. 47) is used, which matches the value of TOF at $k = 20$ and $k = 100$ and allows $\tilde{k}^*$ to approach infinity as T^* goes to zero. Table 4 lists the initial guess parameters for various cases.

$$\tilde{k}^* = \left[\frac{T_1(T_0 - T^*)10 - T_0\sqrt{20}(T_1 - T^*)}{T^*(T_0 - T_1)}\right]^2 \tag{47}$$

$$T_0 = TOF(20)$$

$$T_1 = TOF(100)$$

To expedite initial guess computation the auxiliary function W can be precomputed for $k = 20$ and $k = 100$, simplifying the expression of TOF at these points as follows:

$$TOF(20) = S\sqrt{1 - 20\tau}\left[\tau + 0.04940968903(1 - 20\tau)\right] \tag{48}$$

$$TOF(100) = S\sqrt{1 - 100\tau}\left[\tau + 0.00999209404(1 - 100\tau)\right]$$

To evaluate the performance of the hyperbolic initial guess, a 1 million case Monte-Carlo Test Run A (see Table 3) is performed. Figure 6 shows the distribution of
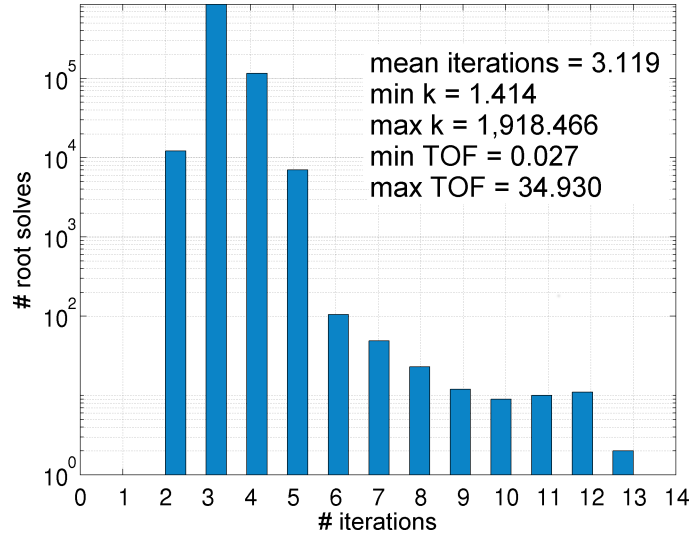
Figure 6: Iteration distribution
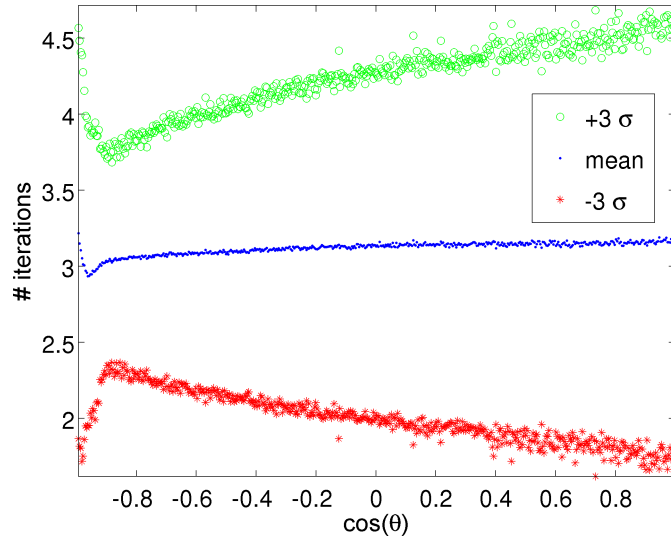
Figure 7: Iteration vs. Transfer angle

number of iterations required corresponding to each root solve. For 99.28% of the cases the method converges in 2 - 4 iterations, with 87.66% of cases taking 3 or fewer iterations. The worst case iteration count (for 1 out of 1 million cases) was 13. Figure 7 shows iteration count statistics as a function of transfer angle where each data point represents the mean and $\pm 3\sigma$ bounds for binned neighboring solutions ordered according to $\cos(\theta)$. Remarkably, the initial guess leads to leads to good performance across the whole range of transfer angle. As shown later, the velocity

Table 5: Zero-rev initial guess parameters (E_1, E_2), using Eq. 44, for $d = \pm 1$

Region	k_n	k_m	k_i	Z	α	F_0	F_1	F_i	F^*
E_1	0	$\sqrt{2}$	$\frac{1}{\sqrt{2}}$	$\frac{1}{2}$	1	$TOF(0)$	T_p	$TOF(\frac{1}{\sqrt{2}})$	T^*
E_2	0	-1	$\frac{-1}{2}$	$\frac{1}{2}$	1	$TOF(0)$	$TOF(-1)$	$TOF(\frac{-1}{2})$	T^*

vector resolution will suffer in accuracy when θ is near $N\pi$, despite that the root-solve procedure is well behaved.

2.5.2 Elliptical zero revolution initial guess generation
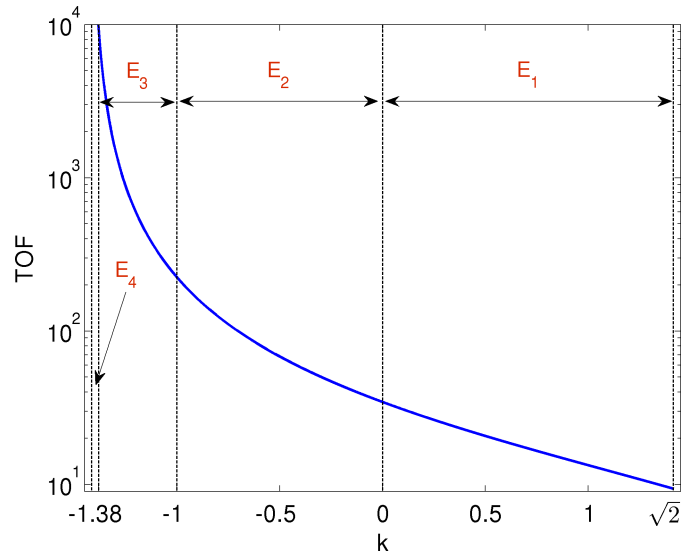
Figure 8: Zero revolution solution space $[r_1 = 1,\ r_2 = 7.098,\ \theta = 69.37\ (\text{deg})]$

Taking into account the curvature changes, the zero revolution solution space is divided into 4 different regions $(E_{1...4})$ as shown in Figure 8. Separate approximations to the TOF function are formulated for each of these regions. Specifically, for regions E_1 and E_2 we use the general initial guess formula given by Eq. 44. For regions E_3 and E_4 we modify Eq. 44 to approximate the inverse of the TOF, thereby making $F(x)$ go to zero as TOF goes to infinity. Also, contrary to the hyperbolic case, the general characteristics of zero-rev solution space are independent of the transfer direction, d.

Table 5 lists the initial guess parameters for the two regions, E_1 and E_2. For regions E_3 and E_4 we modify the the general 3 point interpolation formula (Eq. 44)

to approximate the inverse of the TOF function as follows:

$$\tilde{k}^* = -c_4 \left(\frac{(\gamma_1 c_1 - c_3 \gamma_3)c_2 + c_3 c_1 \gamma_2}{\gamma_3 c_1 - c_3 \gamma_1 - \gamma_2 c_2} \right)^{\frac{1}{\alpha}} \tag{49}$$

$$\gamma_1 = F_i(F^* - F_n)$$

$$\gamma_2 = F^*(F_n - F_i)$$

$$\gamma_3 = F_n(F^* - F_i)$$

Table 6 lists the various initial guess parameters corresponding to Eq. 49, for the regions E_3 and E_4, respectively.

Table 6: Zero-rev initial guess parameters (E_3, E_4), using Eq. 49, for $d = \pm 1$

Region	k_n	k_m	k_i	c_1	c_2	c_3	c_4	α	F_n	F_i	F^*
E_3	-1	$-\sqrt{2}$	-1.38	$\frac{540649}{3125}$	256	1	1	16	$TOF(-1)^{-1}$	$TOF(-1.38)^{-1}$	T^{*-1}
E_4	-1.38	$-\sqrt{2}$	-1.41	$\frac{49267}{27059}$	$\frac{67286}{17897}$	$\frac{2813}{287443}$	$\frac{4439}{3156}$	243	$TOF(-1.38)^{-1}$	$TOF(-1.41)^{-1}$	T^{*-1}

For various intermediate values of k, values for W are precomputed to avoid evaluation at runtime. Equation 50 lists the various intermediate values of W required for the zero-rev initial guess.

$$W(-1.41) = 4839.684497246 \quad , \quad W(-1.38) = 212.087279879 \tag{50}$$

$$W(-1) = 5.712388981 \quad , \quad W\left(\frac{-1}{2}\right) = 1.954946607$$

$$W\left(\frac{1}{\sqrt{2}}\right) = 0.6686397730$$

For $k = 0$ the TOF equation simplifies:

$$TOF(0) = S\left(\frac{\sqrt{2}}{4}\pi + \tau\right) \tag{51}$$

The correct approximation region is identified by comparing T^* with TOF values at $k = 0$, $k = -1$ and $k = -1.138$.

Similar to the hyperbolic case, a 1 million case Monte-Carlo Test Run B (see table 3) is performed for initial guess performance evaluation. Figure 9 shows the distribution of number of iterations corresponding to each root solve. The root solving process converges in 2 - 4 iterations, with a 96.25% cases taking 3 iterations or

33
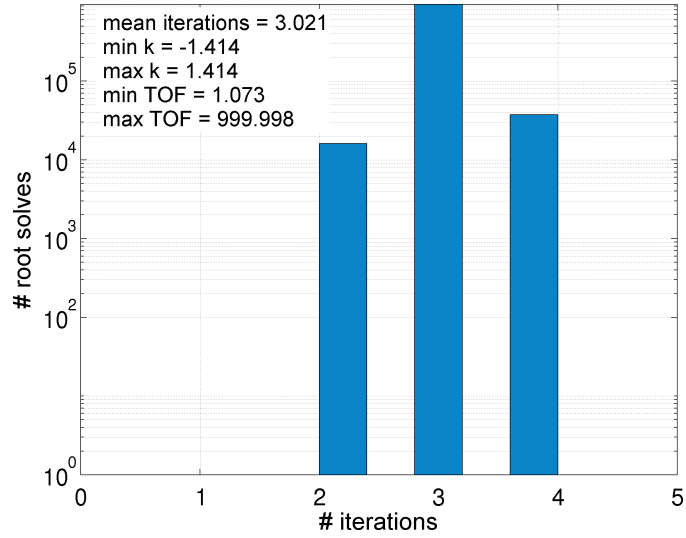
Figure 9: Zero-rev: Iteration distribution
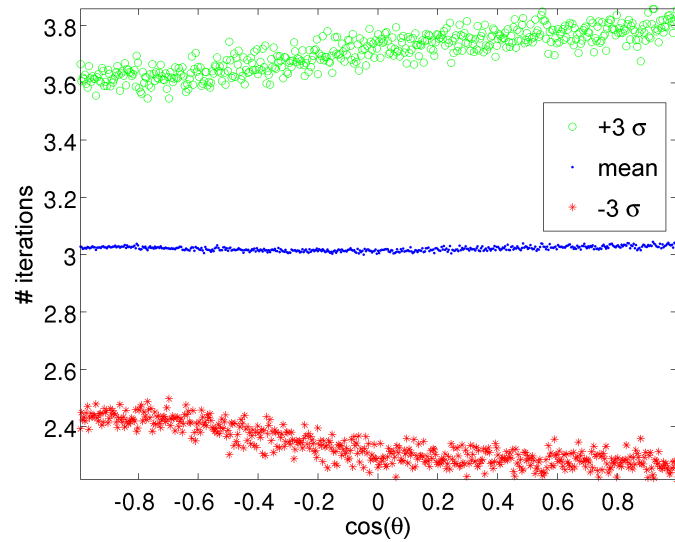
Figure 10: Zero-rev: Iteration vs. Transfer angle

fewer. Figure 10 and 11 bin the results according to transfer angle and flight time, respectively. The initial guess strategy performs well over the whole solution space, with a slight increase in number of iterations for high TOF cases (near the limit of $k = -\sqrt{2}$).
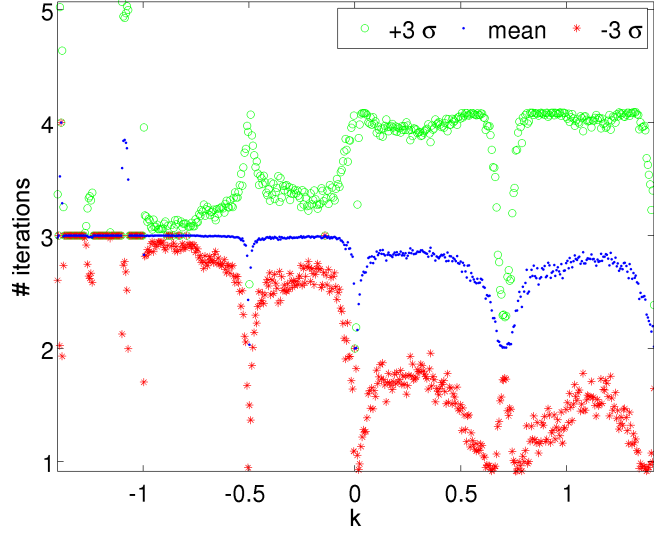
Figure 11: Zero-rev: Iteration vs. k

2.5.3 Multiple revolution initial guess generation

Figure 12 shows the multi-rev solution space for an example i^{th} revolution transfer. The procedure for multi-rev initial guess generation is broken down into two parts, 1) computing $\tilde{k}_{b,i}$, a close approximation to $k_{b,i}$ and 2) using $\tilde{k}_{b,i}$ to compute an approximation of k_i^*.
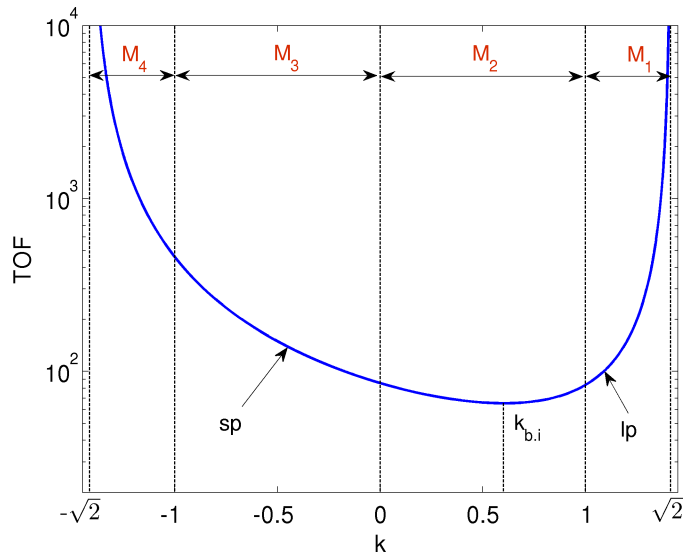
Figure 12: Multi-rev solution space $[r_1 = 1,\ r_2 = 7.098,\ \theta = 69.37\ (\text{deg})]$

2.5.3.1 Approximating $T_{b,i}$

We start by first computing a close approximation to $k_{b,i}$, given by $\tilde{k}_{b,i}$. A close approximation to $k_{b,i}$ helps us in computing the multi-rev initial guess. The slope of the TOF function is zero at $k = k_{b,i}$. Equating the first derivative of the TOF equation (Eq. 37) wrt k to zero, two solutions of τ as a function of N and $k_{b,i}$ are obtained:

$$\tau = \frac{4W'}{4W'k_{b,i} + 3W \pm \sqrt{9W^2 + 8W'}} \tag{52}$$

Using the positive root solution we plot τ as a function of $k_{b,i}$ and associated $\Delta E_{b,i}$ (see Eq. 10) for different values of i in Fig. 14.
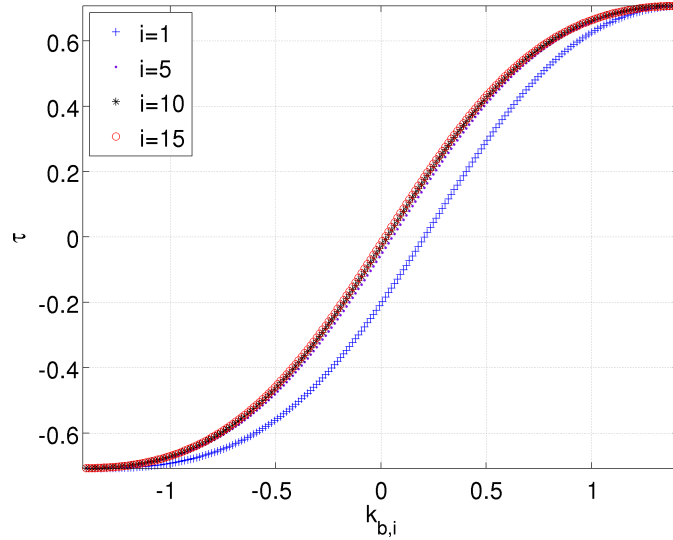
Figure 13: $k_{b,i}$ vs. τ

Remarkably, τ is found to be a well-behaved function of $\Delta E_{b,i}$ or $k_{b,i}$. Figure 15 plots τ vs. $k_{b,i}$ in the region close to $k_{b,i} = -\sqrt{2}$. Given a value of i, all values of $k_{b,i}$ which give $\tau \geq \frac{-1}{\sqrt{2}}$ are permissible. As i approaches infinity the minimum permissible value of $k_{b,i}$ approaches the limit $-\sqrt{2}$. Similarly, in the τ vs. $\Delta E_{b,i}$ plot, the point at which the curve crosses zero ($\Delta E_{b,i0}$) is only a function of i and rapidly approaches π as i goes to infinity. The curve is nearly symmetrical about $\Delta E_{b,i0}$ and this symmetry becomes exact as i approaches infinity. With these insights, an approximation of
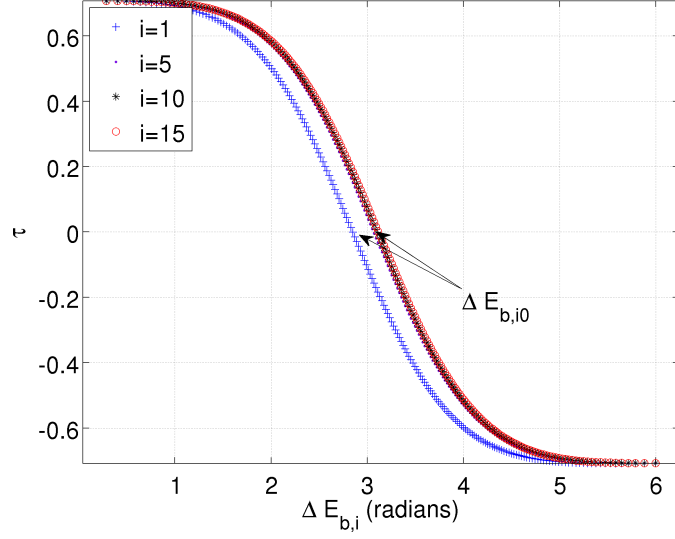
Figure 14: $\Delta E_{b,i}$ vs. τ
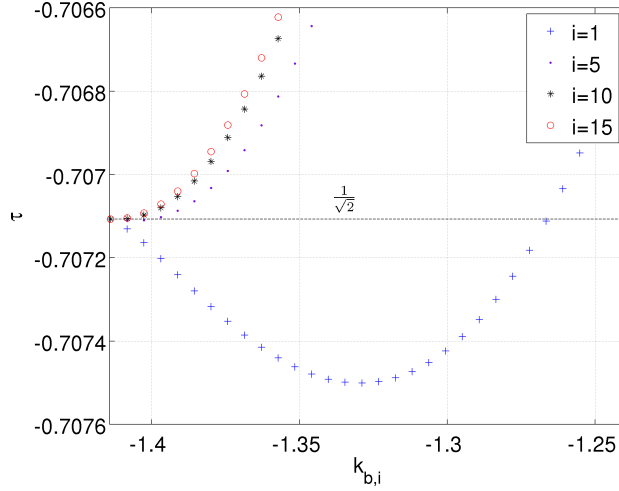
Figure 15: Behaviour of $k_{b,i}$ near $-\sqrt{2}$

$\Delta E_{b,i}$ as function of τ is given by follows:

$$\tilde{\Delta E}_{b,i} = v_2 \left[1 - \text{sgn}(\tau)\right] + v_2 \, \text{sgn}(\tau) \left(\frac{1}{1+v_1}\right)^{1/4} \tag{53}$$

$$v_1 = \frac{8\,|\tau|}{v_2(\sqrt{2} - 2\,|\tau|)}$$

$$v_2 = \Delta E_{b,i0}$$

The inverse of the approximation as defined above, matches the value of τ at $\Delta E_{b,i} = 0$ and $\Delta E_{b,i} = \Delta E_{b,i0}$ and the slope at $\Delta E_{b,i} = 0$. Finally, the transformation

defined in Eq. 10 along with proper sign is used to compute $\tilde{k}_{b,i}$ from $\tilde{\Delta E}_{b,i}$ as follows:

$$\tilde{k}_{b,i} = \text{sgn}(\pi - \tilde{\Delta E}_{b,i})\sqrt{\cos(\tilde{\Delta E}_{b,i}) + 1} \tag{54}$$

For $i = 1 \rightarrow 20$, $\Delta E_{b,i0}$ is precomputed and stored as constant, and for $i > 20$, $\Delta E_{b,i0}$ is set equal to π. The first 20 values of $\Delta E_{b,i0}$ are given in Eq. 55.

$$\Delta E_{b,i0}(1 \rightarrow 10) = [2.848574, 2.969742, 3.019580, 3.046927, 3.064234, 3.076182, 3.084929, 3.091610, 3.096880, 3.101145]$$
$$\tag{55}$$

$$\Delta E_{b,i0}(11 \rightarrow 20) = [3.104666, 3.107623, 3.110142, 3.112312, 3.114203, 3.115864, 3.117335, 3.118646, 3.119824, 3.120886]$$

Figure 16 shows the absolute error in $\tilde{k}_{b,i}$ as a function of τ for four different values of i. The initial guess $\tilde{k}_{b,i}$ performs well over the full range of τ. Figure 17 shows the iteration count distribution for the Test Run C (see Table 3), using Newton's method. A total number of 1,771,749 minimization root solves were performed. One average, 3-4 Newton-Raphson iterations are needed to satisfy the convergence tolerance.

Note that the only purpose of finding the minimum of the multi-rev curve is to robustly bound the root-solve procedure for both the left and right branches. Considering that an accurate approximation for $T_{b,i}$ is now available, the algorithm can actually skip the minimization phase for a large number of cases, while still maintaining robustness. We check solution feasibility by comparing T^* to $\tilde{T}_{b,i}$. A root solve for $k_{b,i}$ is only required if T^* is close to (within some user defined percentage) and less than $\tilde{T}_{b,i}$. This strategy reduces the number of minimization root solves, resulting in a significant increase in performance of the multi-rev algorithm. The above strategy has been verified to work with a user defined percentage of 20 when testing over $\sim$3 billion multi-rev solutions. A mathematically robust strategy for bounding the root-solves without performing a full minimization is developed in the next section. In general, we are guided by the basic principle that the minimization step is expensive and should be avoided when possible.
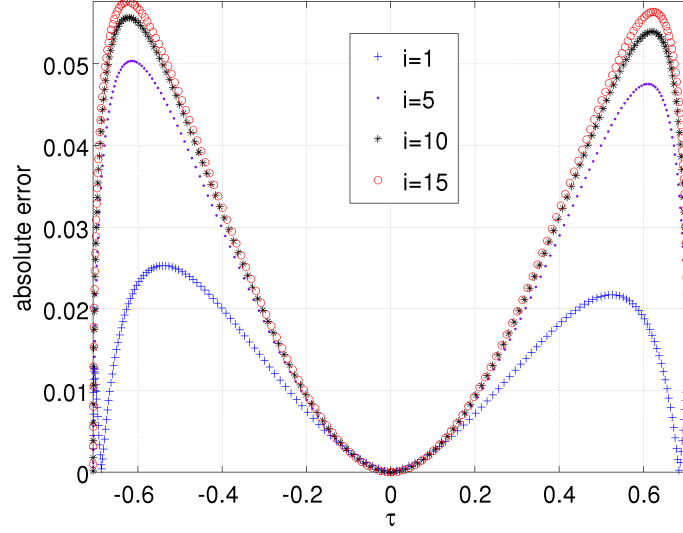
Figure 16: Absolute error in $k_{b,i}$
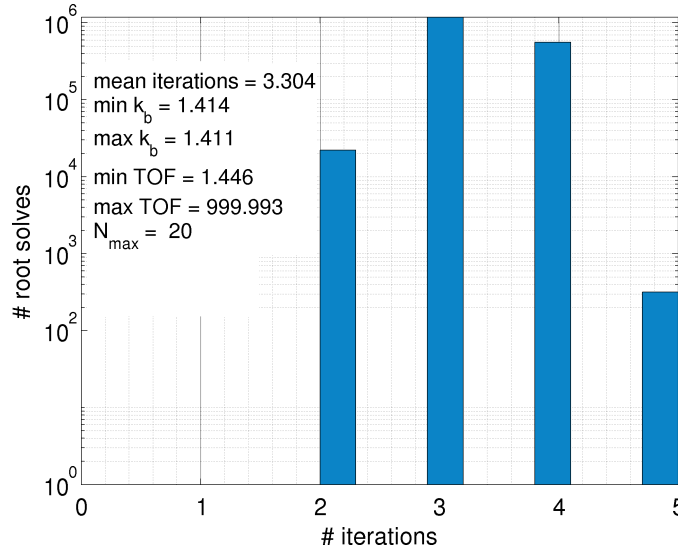
Figure 17: Iterations distribution: minimization phase

2.5.3.2 *Multi-rev initial guess:*

To generate an approximation of k_i^*, the multi-rev solution space is divided into 4 regions as shown in Fig. 12. For a given value of d, T^* and i, there are two possible solutions of k^*. The branch to the right of the $k = k_{b,i}$ denotes the long period (lp) solution and the branch on the left denotes the short period (sp) solution. The bottom most point $(k_{b,i})$ can either be positive or negative and is approximated by

$\tilde{k}_{b,i}$, with a corresponding TOF approximation, $\tilde{T}_{b,i}$.

For a fixed sign of $\tilde{k}_{b,i}$ and depending upon the value of T^* and $\tilde{T}_{b,i}$, six possible initial guess regions (3 for each, sp and lp) are identified. The general 3 point approximation formula defined by Eqs. 44 and 45 is used for computing the initial guess for all of the 6 cases.

Table 7 lists the 6 six cases along with their initial guess parameters for $\tilde{k}_{b,i} \geq 0$. For cases with T^* in M_1 or M_4 regions , the inverse of the TOF function is used to compute the initial guess. In a similar fashion Table 8 lists the 6 cases and their associated initial guess parameters for $\tilde{k}_{b,i} \leq 0$.

Table 7: Multi-rev initial guess parameters ($\tilde{k}_{bi} \geq 0$), used in Eq. 44

Sol. Type	T^* location	$\tilde{k}_{b,i}$ location	k_n	k_m	k_i	z	α	F_0	F_1	F_i	F^*
lp	M_1	M_1	$\tilde{k}_b$	$\sqrt{2}$	$\frac{\tilde{k}_b+\sqrt{2}}{2}$	$\frac{1}{4}$	2	$\tilde{T}_b^{-1}$	0	$TOF(k_i)^{-1}$	T^{*-1}
lp	M_1	M_2	1	$\sqrt{2}$	$\frac{1+2\sqrt{2}}{3}$	$\frac{4}{9}$	2	$TOF(1)^{-1}$	0	$TOF(k_i)^{-1}$	T^{*-1}
lp	M_2	M_2	$\tilde{k}_b$	1	$\frac{1+\tilde{k}_b}{2}$	$\frac{1}{4}$	2	$\tilde{T}_b$	$TOF(1)$	$TOF(k_i)$	T^*
sp	M_1 or M_2	M_1 or M_2	0	$\tilde{k}_b$	$\frac{\tilde{k}_b}{2}$	$(\frac{1}{2})^\alpha$	$\frac{6}{5}$	$TOF(0)$	$\tilde{T}_b$	$TOF(k_i)$	T^*
sp	M_3	M_1 or M_2	-1	0	$\frac{-1}{2}$	$\frac{1}{2}$	1	$TOF(-1)$	$TOF(0)$	$TOF(\frac{-1}{2})$	T^*
sp	M_4	M_1 or M_2	-1	$-\sqrt{2}$	$\frac{-1-2\sqrt{2}}{3}$	$\frac{4}{9}$	2	$TOF(-1)^{-1}$	0	$TOF(k_i)^{-1}$	T^{*-1}

Analogous to the previous sections, W for the multi-rev case can be precomputed (Table 9) for intermediate values of k to expedite initial guess generation.

As $\tilde{k}_{b,i}$ is only an approximation for $k_{b,i}$, there exists ambiguity in successfully

Table 8: Multi-rev initial guess parameters ($\tilde{k}_{b,i} \leq 0$), used in Eq. 44

Sol. Type	T^* location	$\tilde{k}_{b,i}$ location	k_n	k_m	k_i	z	α	F_0	F_1	F_i	F^*
sp	M_4	M_4	$\tilde{k}_b$	$-\sqrt{2}$	$\frac{\tilde{k}_b-\sqrt{2}}{2}$	$\frac{1}{4}$	2	$\tilde{T}_b^{-1}$	0	$TOF(k_i)^{-1}$	T^{*-1}
sp	M_4	M_3	-1	$-\sqrt{2}$	$\frac{-1-2\sqrt{2}}{3}$	$\frac{4}{9}$	2	$TOF(-1)^{-1}$	0	$TOF(k_i)^{-1}$	T^{*-1}
sp	M_3	M_3	$\tilde{k}_b$	-1	$\frac{-1+\tilde{k}_b}{2}$	$\frac{1}{4}$	2	$\tilde{T}_b$	$TOF(-1)$	$TOF(k_i)$	T^*
lp	M_3 or M_4	M_3 or M_4	$\tilde{k}_b$	0	$\frac{\tilde{k}_b}{2}$	$\frac{1}{4}$	2	$\tilde{T}_b$	$TOF(0)$	$TOF(k_i)$	T^*
lp	M_2	M_3 or M_4	0	1	$\frac{1}{2}$	$(\frac{1}{2})^\alpha$	$\frac{6}{5}$	$TOF(0)$	$TOF(1)$	$TOF(\frac{1}{2})$	T^*
lp	M_1	M_3 or M_4	1	$\sqrt{2}$	$\frac{1+2\sqrt{2}}{3}$	$\frac{4}{9}$	2	$TOF(1)^{-1}$	0	$TOF(k_i)^{-1}$	T^{*-1}

Table 9: Precomputed W values

k	W
$\frac{-1-2\sqrt{2}}{3}$	27.25239909 + 27.75304668N
-1	5.71238898 + 2πN
$\frac{-1}{2}$	1.95494660 + 2.71408094N
0	$\frac{\sqrt{2}}{4}(\pi + 2\pi N)$
$\frac{1}{2}$	0.75913433 + 2.71408094N
1	0.57079632 +2πN
$\frac{1+2\sqrt{2}}{3}$	0.50064759+27.75304668N
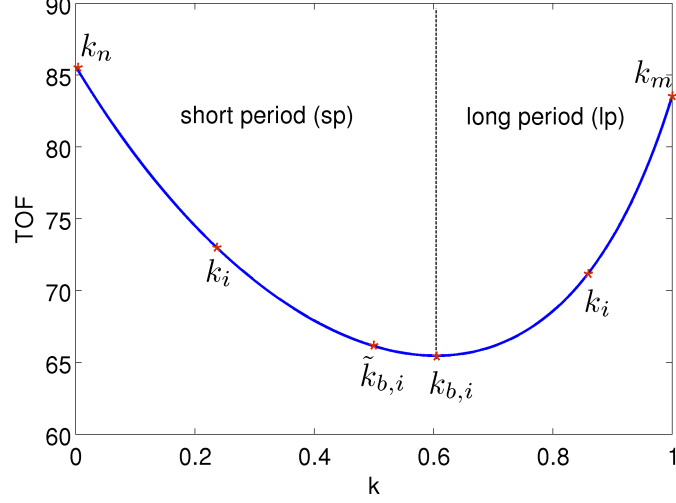
Figure 18: Bounding minimization root solve

bounding the root solve, whenever $\tilde{k}_{b,i}$ is one of the region bounds. Figure. 18 depicts the case when T^* lies in the region M_2 with $\tilde{k}_{b,i} > 0$. The algorithm first starts by computing k_i for both the branches. Next, consider the left branch (sp solution) in Fig. 18 if $TOF(k_i) < T^*$ the and $TOF(k_i) > \tilde{T}_{b,i}$ then a root between k_i and k_n is guaranteed. Similarly, for the right branch (lp solution) if $TOF(k_i) < T^*$ and $TOF(k_i) > \tilde{T}_{b,i}$ then we can guarantee a root between k_i and k_m.

Assuming, $T^* \geq \tilde{T}_{b,i}$, ambiguity arises when $T^* < TOF(k_i)$ in either of the branches. In such a case, the algorithm first locates the branch on which $\tilde{k}_{b,i}$ lies by computing the sign of the slope at $k = \tilde{k}_{b,i}$. In Fig. 18, the left branch constitutes $\tilde{k}_{b,i}$ and can be bounded, with a guaranteed root between $\tilde{k}_{b,i}$ and k_n. For the right ambiguous branch, the root solver at each iteration checks the validity of the current step by checking sign of the slope at each iteration. If the computed slope at a given iteration has the wrong sign, a new approximation of $k_{b,i}$ is computed by performing Newton step updates, noting that the derivatives TOF' and TOF'' at the current k are already present. The partial minimization root solve is performed until one out of following two conditions is satisfied: 1) the root solver can be bounded, with the new approximation of $k_{b,i}$ having the correct sign of the slope or 2) $k_{b,i}$ is located

to within the user defined convergence tolerance. Having computed a new bound, the initial guess is recalculated and the root solver proceeds to find the solution. In the case when $k_{b,i} \approx 0$, the approximation $\tilde{k}_{b,i}$ may have the the wrong sign. As the initial guess parameter set (see Tables 7 and 8) changes depending on the sign of $\tilde{k}_{b,i}$, a minimization root solve is also performed whenever $TOF(0) < \tilde{T}_{b,i}$. The proposed strategy of computing partial minimization root solves on the fly, robustly bounds both the lp and sp solution branches while keeping the number of minimization root solve iterations to a minimum.

mean iterations = 3.011
min k = -1.414
max k = 1.414
min TOF = 1.341
max TOF = 1999.996
N_{max} = 20
root solves = 26,530,092
full minimizations = 585,196
partial minimizations = 5,466
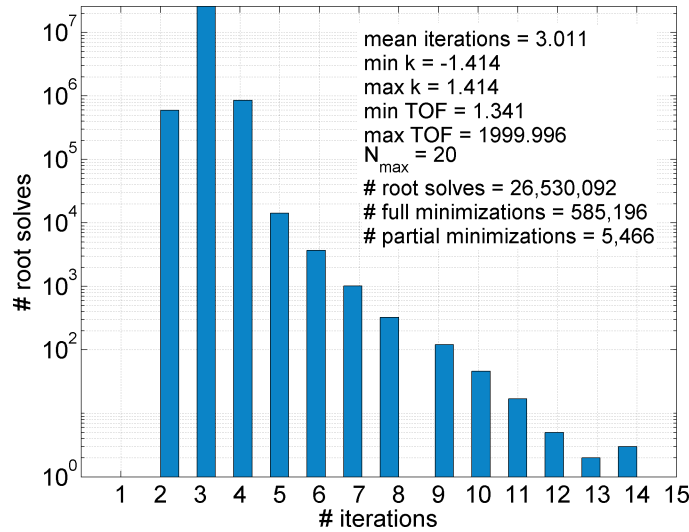
Figure 19: Multi-rev: Iteration count distribution

To gauge the performance of the multi-rev initial guess, a 1 million case monte-carlo Test Run D (see table 3) is performed (see Fig.19-21). The value N_{max} is set to 20, resulting in 26,530,092 root solves (with zero failures) out of which 25,500,957 (or 98.01%) cases take 3 iterations or less to converge (see Fig. 19). There are 7 that failed to converge to the specified 1E-13 tolerance. These cases show no improvement in their residual if more 10 iterations are performed. The maximum residual for these 7 partially converged cases is found to be 2.043E-13.
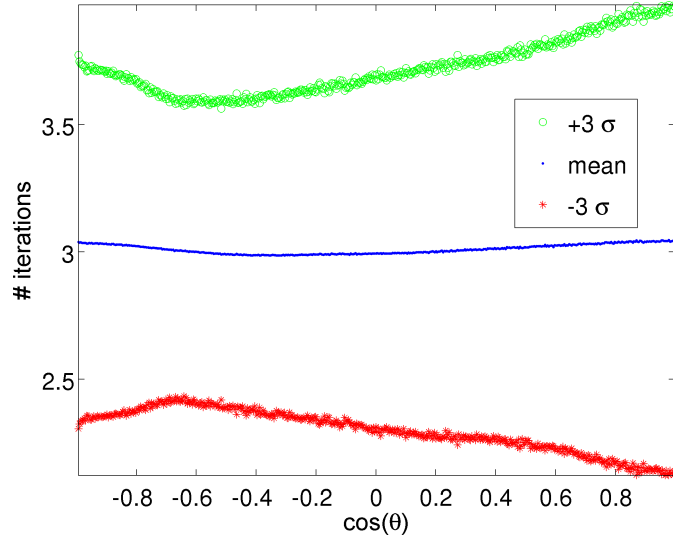
Figure 20: Multi-rev: Iteration vs. Transfer angle
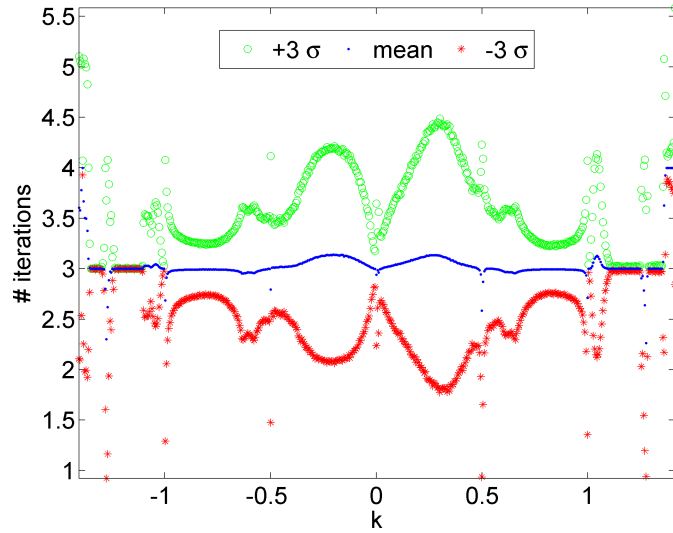
Figure 21: Multi-rev: Iteration vs. k

2.6   Implementation Details

The proposed algorithm to solve the Lambert problem, is implemented keeping run-time speed and numerical robustness in mind.

For the hyperbolic case, a $\sim 15\%$ increase speed is observed by replacing inverse hyperbolic cosine (required for computing W) with its equivalent logarithmic form, defined by Eq. 56. This logarithmic form of inverse hyperbolic cosine is used for both

the initial guess generation and the hyperbolic root solves.

$$\cosh^{-1}(v) = \log(v + \sqrt{v^2 - 1}) \tag{56}$$

$$v = k^2 - 1$$

For the elliptical initial guess generation, computing various intermediate TOF values requires evaluation of the expensive inverse cosine function. As high accuracy in the initial guess is not required, we approximate the inverse cosine functions by a minimax approximation, computed via Maple, accurate up to 5-6 digits, given by Eq. 57.

$$\cos^{-1}(x) \approx \frac{\pi}{2} - \mathrm{sgn}(x) \begin{pmatrix} \frac{0.000014773722+(1.1782782-0.52020038\,|x|)|x|}{1.1793469+(-0.53277664-0.14454764\,|x|)|x|} & 0 \leq |x| \leq 0.6 \\ \frac{0.011101554+(8.9810074+(-14.816468+5.9249913\,|x|)|x|)|x|}{9.2299851+(-16.001036+6.8381053\,|x|)|x|} & 0.6 < |x| \leq 0.97 \\ \frac{-35.750586+(107.24325-70.780244\,|x|)|x|}{27.105764-26.638535\,|x|} & 0.97 < |x| < 0.99 \\ \sin^{-1}(|x|) & |x| \geq 0.99 \end{pmatrix} \tag{57}$$

During testing, it was found that for $k \approx 0$, W is computed accurately to only up to $\sim$13 digits in double precision arithmetic. This precision loss increases the number of root solve iterations required from 3 to up to 7 in some rare cases. The impact on speed from this precision loss is insignificant as k is seldom very close to zero. Nevertheless, to prevent this precision loss, W is computed via a series when $|k| < 1E-3$:

$$W = \left(\frac{2N\pi + \pi}{4}\right)\sqrt{2} - k + \frac{3(2N\pi + \pi)}{16}\sqrt{2}k^2 - \frac{2}{3}k^3 + \frac{15}{128}(2N\pi + \pi)\sqrt{2}k^4 - \frac{2}{5}k^5 + O(k^6) \tag{58}$$

Note that, this precision loss is found to only affects the number iterations when the root solver tolerance is set to 3E-13 or less.

2.7 Performance Comparison

In this section the performance of the proposed Lambert formulation is evaluated against the well known Gooding's Method.[48] Previous comparative studies have found Godding's method to be among the fastest and most accurate of the methods available.[98,70] Note that Gooding's method always performs 3 iterative corrections

44

using Halley's iteration method with well tuned initial guesses, and hence has a fixed computation cost per root solve iteration. On the other hand the current formulation subjects the root solver to a normalized tolerance of 1E-13 and a maximum iteration count of 20. The original code from Gooding is taken directly from Ref.[48] Both Gooding's code and the new formulation is compiled using the Intel Fortran compiler version 12.0 subject to "O2" speed optimization flag. The test hardware used during the performance comparison is comprised of an Intel Xeon X5650 CPU @ 2.67 Ghz , running on X86_64 Linux workstation.

2.7.1 Accuracy

To compare the absolute accuracies of both the methods, 10,677,126 "truth" solutions (see Test Run E in Table 3) are generated using a Kepler solver, in quadruple precision, subject to a tolerance of 1E-28. The final solution (after the random initial conditions as specified in Table 3) set has a minimum and maximum θ of 0.076 degrees and 179.94 degrees, respectively. The resulting minimum and maximum value of eccentricity (e) are 1E-3 and 268.75, respectively. Hence, the test suite is deemed exhaustive enough to capture the global accuracy behavior of both the solvers.
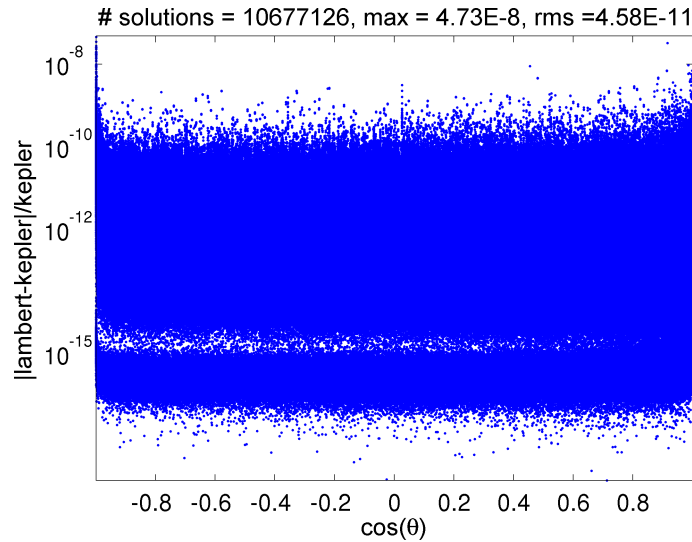
Figure 22: Gooding's method relative error vs. transfer angle

45

solutions = 10677126, max = 4.72E-8, rms =4.56E-11
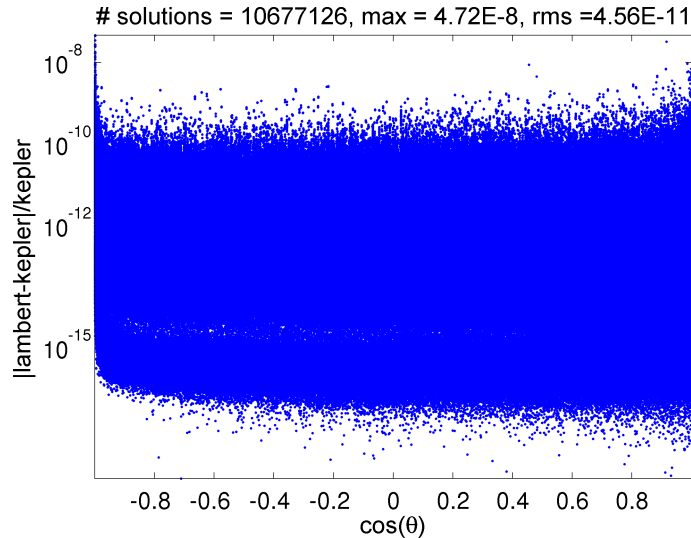
Figure 23: New Lambert relative error vs. transfer angle

Figures 22 and 23 shows the relative errors between the Kepler solution and the two Lambert implementations as a function of $\cos(\theta)$. Both methods show a similar accuracy profile, with almost the same RMS and max relative errors when compared to the truth. Both methods suffer significant loss of accuracy with increase in relative error by almost 1.5 orders of magnitude for θ close to 0, π or 2π.

To differentiate between the elliptical and hyperbolic case, the relative errors are plotted as a function of eccentricity in Figs. 24 and 25. Unlike, Gooding's method, our algorithm stops once the root solve tolerance is met, hence may require less than 3 iterations to converge in some cases. Both solvers are found to experience numerical degradation as eccentricity approaches unity, while they both perform well in the hyperbolic solution regime.

The exhaustive accuracy comparison performed here demonstrates that the absolute accuracy of the proposed Lambert formulation is statistically similar to that of Gooding's method. As stated earlier, Gooding's method is considered to be one the most accurate Lambert solvers available.[70, 98]
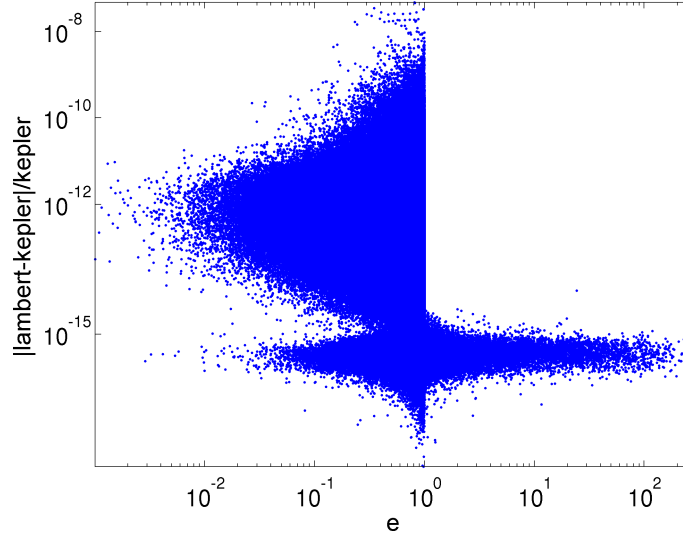
Figure 24: Gooding's method relative error vs. e
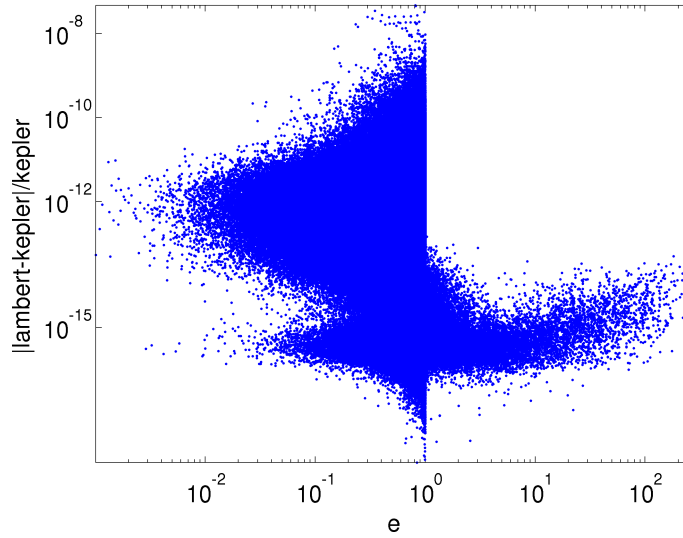
Figure 25: Current method relative error vs. e

2.7.2 Runtime Comparisons

Figure 26 shows the variation of speedup over the Gooding method as a function of $\frac{TOF}{T_p}$. Each data point represents one million Lambert cases with input vectors $\vec{r_1}$ and $\vec{r_2}$ varying according to Test Run B (see Table 3). The ratio $\frac{TOF}{T_p}$ is varied from 0.05 to 1,000, there-by allowing both hyperbolic and elliptical multi-rev solutions. For the hyperbolic case the speedup varies from 1.28 to 4.25, with the mean and mean absolute deviation being 1.87 and 0.58, respectively. The runtime for the Gooding method

47

increases as the $\frac{TOF}{T_p}$ approaches unity. For the elliptical zero-rev case ($N_{max} = 0$), the speedup varies between 7.08 to 1.44, with a mean value and mean absolute deviation of 1.75 and 0.22 , respectively. The zero-rev runtime of the new method gradually increases as TOF increases, due to increase in the iteration count. Gooding's method on the other hand always takes three iterations (without guaranteeing convergence), hence has a fixed evaluation cost.

Table 10: Speedup statistics for various transfers in Fig. 26

Transfer type	N_{max}	Mean speedup	Mean absolute deviation
Hyperbolic	0	1.87	0.58
Elliptical	0	1.75	0.22
Elliptical	5	2.13	0.45
Elliptical	10	2.15	0.45
Elliptical	15	2.16	0.44

For the multi-rev case, a speedup of ~11 times is achieved when the ratio $\frac{TOF}{T_p}$ is close to unity. For low TOF cases, we get approximately 2.5 times speedup, as we skip most of the bottom minimizations required to check for multi-rev solution feasibility. At a certain TOF value, the number multi-rev of solutions become significant and speedup peaks near 2 when N_{ub} starts to overtake N_{max}. Further increasing the ratio $\frac{TOF}{T_p}$ leads to gradual drop in speedup as the number of iterations increases. Table 10 summarizes the mean speedup and mean absolute deviation (MAD) for various cases in Fig. 26.

Due to fact that Gooding's method always takes 3 iterations it cannot guarantee convergence to a specified tolerance value. For some extreme cases with high TOF values, it is found that Gooding's method was not fully converged after 3 iterations. On the other hand the current Lambert formulation implementation nominally includes a variable tolerance root-solver. This slight advantage in robustness comes with a slight disadvantage when computing speedup values. Nevertheless, the proposed method is faster by a factor of ~1.3 to ~3 for most cases considered, where the specific speedup depends on a variety of parameters.

48
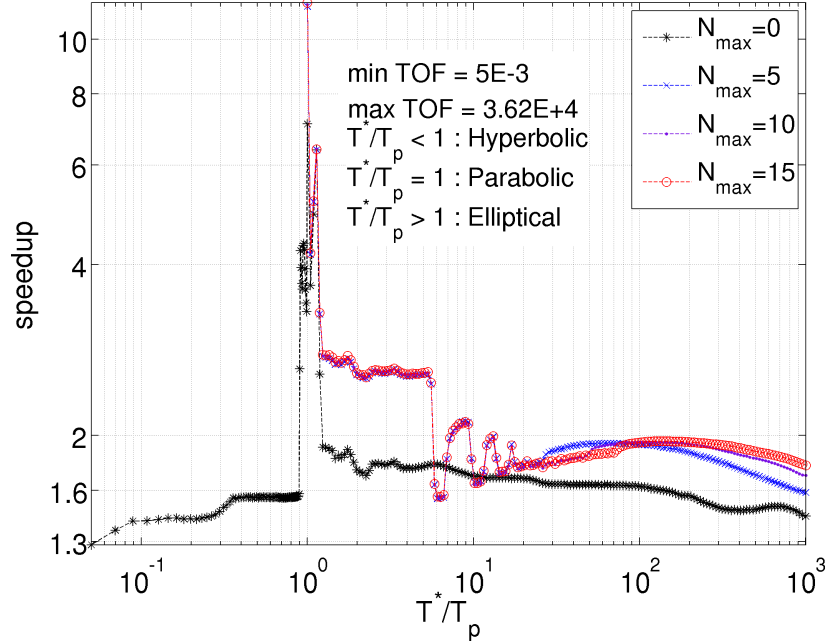
Figure 26: Speedup vs. $\frac{TOF}{T_p}$

2.8   Chapter Conclusion

A universal, multi-rev Lambert formulation based on the cosine of the change in eccentric anomaly is proposed in this chapter. The new universal variable "k" is introduced, resulting in a simplified form of the TOF equation. The formulation requires only a single transcendental function evaluation and enjoys simple derivative expressions. An accurate initial guess strategy is presented based on rational polynomial approximations. The universal variable corresponding to the minimum multi-rev TOF is found to be a well-behaved function of a geometry based parameter. Accurate approximations of the inverse function leads to a fast and robust bounding strategy for root-solving both sides of the multiple revolution curve. The initial guess strategy, coupled with Halley's iteration method leads to convergence in 2 or 3 iterations (with tolerance set to 1E-13) for a vast majority of the test cases.

The new method and its implementation is on par in term of accuracy, and is typically ∼1.3 to ∼3 times faster when compared to the state of the art Gooding's method. The runtime Fortran code is embedded as part of this thesis in appendix E.

CHAPTER III

HIGH-FIDELITY GEOPOTENTIAL COMPUTATION

3.1 Chapter Summary

The current chapter focuses on improving runtime performance of the gravity geopotential calculations and the associated gradients. A high-fidelity interpolation method (which trades higher memory footprints for faster runtimes) is presented for approximating a scalar quantity and associated gradients in the global 3D domain external to a sphere. The new "Fetch" interpolation model modifies and utilizes the weighting function method originally proposed by Junkins et al. to achieve continuity and smoothness. An overlapping grid strategy ensures a singularity-free domain, while minimizing associated memory costs. Local interpolating functions are judiciously chosen with a new adaptive, order-based selection of local polynomials which minimizes coefficient storage subject to a radially mapped residual tolerance. Analytic inversions of the normal equations associated with each candidate polynomial allow for rapid solutions to the least squares process without resorting to the conventional numerical linear system solvers. The gradient and higher order partial derivatives are computed directly with no memory cost, and are smooth and continuous to a user-specified order. The method is specifically applied to interpolate the GRACE GGM03C geopotential model up to degree and order 360. Highly tuned interpolation models of various resolutions are presented and discussed in detail. Released Fetch interpolation models of the geopotential include resolutions of 33×33, 70×70, 156×156 and 360×360. The memory requirements span from 120 MB to 2360 MB, and the expected speedups over spherical harmonics evaluations span from ∼10 to ∼3,000 fold. The break even resolution for runtime speeds is approximately degree and order

8. The models are globally continuous to order 3 and thus may be of interest to a variety of science and engineering applications.

3.2 Chapter Nomenclature

δ	=	Polar angle (0 to 180 degrees)
λ	=	Azimuth (equivalent to geocentric longitude, 0 to 360 degrees)
ϕ	=	Geocentric latitude (δ - 90)
δ, λ, r	=	Spherical coordinate set ($r \geq 1$)
σ	=	Total loop counter
N	=	Total number of coefficients of a general node polynomial
$N_{i,j,k}$	=	Total number of coefficients of a node polynomial, associated to the i, j, k cell node
d	=	Degree of a local node polynomial
m	=	Total number of measurements per cell in each direction
$\boldsymbol{B}$	=	Least square inverse matrix
$\hat{e}_1, \hat{e}_2, \hat{e}_3$	=	Unit vectors in polar angle, azimuth and radial direction
$\eta_{0...4}$	=	User controlled model residual tolerance constants
$\vec{x}$	=	Absolute position vector of the evaluation point in spherical coordinates
$\vec{y}$	=	Normalized (-1 to 1) position vector of the evaluation point in spherical coordinates
$\vec{x}_{i,j,k}$	=	Absolute position vector of the i, j, k cell node in spherical coordinates
$\vec{x}_w$	=	Normalized position vector in spherical coordinates of the evaluation point in the cell domain
Q_d	=	Major candidate polynomial of degree d
$Q_{d,z}$	=	Minor candidate polynomial of degree d and index z
N_{max}	=	Maximum number of coefficients for a node polynomial
O_{max}	=	Maximum degree of a node polynomial
$w_{i,j,k}$	=	Weight function associated to the i, j, k cell node
R_e	=	Mean radius of the Earth
U_F	=	Final composite potential computed using a weighted average
$U_{i,j,k}$	=	Local polynomial associated to the i, j, k cell node
$\vec{C}$	=	Coefficient vector for a node polynomial
$\boldsymbol{D}_{\alpha,\beta,\gamma}$	=	Node polynomial normalization matrix
$\boldsymbol{D}_w$	=	Weight function normalization matrix
Ω	=	Hermite weighting function
SH	=	Spherical Harmonics
MPI	=	Message passing interface

3.3 Introduction and Background

High-fidelity trajectory computation using conventional spherical harmonics (SH) gravity fields is computationally slow and difficult to implement if starting from

scratch.[101, 27, 85] New high order gravity models[132, 55] and growing number of objects in the space catalog[56] is driving the need for a new class of gravity models that enjoy the robustness and flexibility of spherical harmonics while still achieving orders of magnitude in speedup.

Over the years, two general class of alternative methods have been proposed 1) Discrete mass models[71, 144, 107, 106, 88, 145] and 2) Interpolation models. Discrete mass models are often used to provide increased local resolution when used along with spherical harmonics. They are easy to implement and recent studies show that show these models adapt well to the problem of small bodies.[96] A recent study demonstrates these models to provide an order of magnitude in speed improvements using common Graphics Processing Unit (GPU).[112]

The interpolation models on the other hand classicaly trade memory for speed and are becoming increasingly more relevant given the extraordinary memory resources of common computers. Primarily motivated by Junkins, a variety of techniques and basis functions (for interpolating) have been proposed over the years including weighting functions,[69, 68] wavelets,[19] splines,[19, 81] octrees,[28] psuedocenters[59] and 3D digital modeling.[94] Recently the first modern global model, called the cubed sphere model has been published.[66] The highest resolution cubed sphere model of SH degree and order 150 achieves 30 fold speedup over SH and requires 856 MB of storage. The model suffers from small discontinuities across shell boundaries, and requires the storage of extra coefficients for computing acceleration and higher order derivatives. Each interpolation based method in general balances accuracy with efforts to minimize runtime speed and memory footprint while achieving exactness, continuity and smoothness as appropriate.

In this chapter we propose a new hybrid model (called Fetch) which attempts to take advantage of all the previous models, while finding innovative solutions to correct their respective problems. Four main priorities are defined which guide the selection of

the adopted solution method and can be summarized as 1) continuity and smoothness across the global domain to an arbitrary order of derivatives, 2) adaptivity in terms of local vs. global resolution 3) a residual error profile in the noise of the accuracy of the underlying base model (SH in this study), and 4) a non singular approach. All previously proposed gravity field interpolation models fell short when evaluated against each of these ambitious priorities.

Continuity in zeroth and higher order derivatives is a desirable feature for any force model. Global continuity in the first derivative is necessary to accurately represent a conservative field, while continuity in higher order derivatives may be important, depending on the specific application. Most 3D interpolation based methods are not globally continuous or require complex algorithms to achieve continuity. The tricubic interpolation method by Lekien and Marsden[81] is an example that achieves first order continuity but at the expense of reduced accuracy and performance. Furthermore, in their method the interpolants must be of the same degree globally, and accurate high order derivatives of the fitting data are required for the coefficient generation process. The Cubed-Sphere model is discontinuous even to the zeroth order across the shell boundaries, although the discontinuity is small in the published models. Also, like the tricubic method, the degree of the interpolants for the Cubed-Sphere model are fixed globally. The new Fetch model takes advantage of a modified weighted interpolation scheme (the original one developed by Junkins et al.[69]) to achieve third order global continuity.

The classic singularity and associated numerical problems near the the poles (when converting from Spherical to Cartesian frame) is avoided in the Fetch model using a two level overlapping global grid structure with an additional weighting function to ensure continuity between the grids. This approach to remove the singularity only requires extra coefficients in the overlapping region($\sim 0.2\%$ to 5% of the domain

for high and low resolution models, respectively), and is therefore more memory efficient than storing an extra term throughout the entire domain. The new model also introduces a novel polynomial selection strategy based on an ordered listing of potential polynomial basis functions, leading to significant improvements in terms of memory efficiency. The method utilizes an algebraic manipulator to produce high order analytic inverses for the resulting least squares problems to ensure accurate and rapid coefficient evaluation. High order polynomial approximations are feasible via a master-worker implementation of the parallel coefficient generation algorithm (implemented using MPI) drive by a least square fitting algorithm on a non-uniform grid. The original Junkins weighting function method is modified to handle this new memory-saving, non-uniform grid. The interpolated geopotential is *exact* in the sense that accelerations and higher order derivatives are calculated as exact derivatives of the interpolated function. Furthermore, the memory requirements scale approximately linearly with degree of the base field while the speedups are nearly a cubic in the same argument. This favorable memory scaling makes the new Fetch model attractive for use with high order gravity fields.

In this chapter the Fetch interpolation approach is applied to the geopotential application. Information on how to use the Fetch gravity model is given which is followed by a comprehensive performance profiling via a direct comparison with a state-of-the-art, singularity-free SH implementation. The Fetch algorithm is developed to be general in the sense so that a user can control the coefficient generation trade of memory vs. speedup with a minimum number of parameters. In this thesis, the Fetch model is applied to approximate the GRACE GGM03C gravity field. In this thesis, four Fetch models are computed and the qualitative results governing them are presented. Specifically, the highest resolution Fetch model corresponds to the complete 360×360 GMM03C SH field and required on the order of 12,000 CPU hours to compute all of its coefficients. The highly optimized memory burden of

this model is 2.36 GB, leading to as much as three orders of magnitude in runtime speedups over the GGM03C SH model. On the other end of the spectrum, an 8×8 Fetch model is found to approximately match the runtime speed of its associated SH counterpart. The next section of this chapter gives an overview of the Fetch gravity model.

3.4 Fetch Gravity Model Overview

Localized representation of the gravity field follows naturally from the fact that there are uneven gravity undulations over the Earth surface. In order to separate the dominant global effects from the smaller local undulations we formulate the geopotential approximation as given by Eq. 59:

$$U \simeq U_{J_2} + U_F \tag{59}$$

where U represents the total potential from a specified degree and order of the GGM03C spherical harmonics field, U_{J_2} represents the potential only due to the J_2 term, and U_F is the interpolated local potential obtained from the Fetch model. The J_2 term in Eq. 59 is three orders of magnitude more significant than all the higher order terms combined. Removing of U_{J_2} (given by Eq. 60) from higher order terms provides an extra 3-4 digits of accuracy in the potential interpolation at an almost negligible computational cost. Most previous gravity field interpolation efforts have also exploited the benefit of removing the low frequency terms.[67, 59, 28, 66, 112, 19, 140] If applicable, tides or any other time dependent low frequency terms could also be removed from the static interpolation. In this study, however, only the mean J_2 term is removed:

$$U_{J_2} = -\frac{\mu}{r} \left[J_2(\frac{R_e}{r})^2 P_2(\sin(\phi)) \right] \tag{60}$$

where μ and R_e are the reference gravitational parameter and radius of the Earth respectively, r and ϕ are the magnitude and geocentric latitude of the position vector

respectively, and P_2 is the second degree Legendre polynomial. The U_F term in Eq. 59 represents the final composite polynomial, and is computed using a weighted average of the eight node polynomials whose centers depend on the spacing of neighboring 8 cells. (see Fig. 27 and Eq. 61).[69,67] The coefficients for each of the eight local node polynomials are computed via least squares fits of the local geopotential (with J_2 removed). More details are provided later on the least squares solutions and candidate forms for the local node polynomials.

Evaluation point (★) inside a single CELL given by $\vec{x} = [x_1, x_2, x_3]^T$, surrounded by 8 cell nodes (+) at each of the cell corners.

Each box within the global domain is a 'cell'. Each box vertex (+) is a cell 'node.' Each node has a corresponding 'node polynomial' that is valid in each of the eight node-touching cells. The final interpolation is a weighted average of the 8 node polynomials associated to each cell.
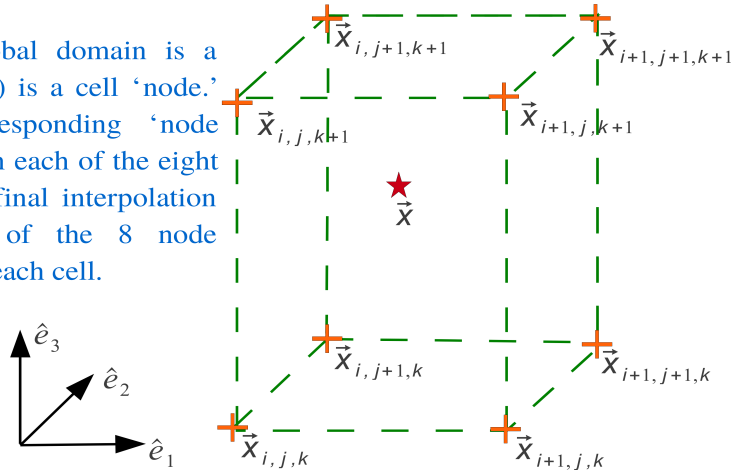
Figure 27: Cell and node geometry for the weighed interpolation

56

$$U_F = \sum_{\alpha=i}^{i+1} \sum_{\beta=j}^{j+1} \sum_{\gamma=k}^{k+1} w_{\alpha,\beta,\gamma}(\vec{x}_w) U_{\alpha,\beta,\gamma}(\vec{y})$$

$$\vec{x}_w = \mathbf{D}_w^{-1}(\vec{x} - \vec{x}_{i,j,k})$$

$$\vec{y} \equiv \vec{x}_{u_{\alpha,\beta,\gamma}} = \mathbf{D}_{\alpha,\beta,\gamma}^{-1}(\vec{x} - \vec{x}_{p_{\alpha,\beta,\gamma}})$$

$$\vec{x}_{p_{\alpha,\beta,\gamma}} = \left[\frac{(\vec{x}_{\alpha+1,\beta,\gamma}+\vec{x}_{\alpha-1,\beta,\gamma})\cdot\hat{e}_1}{2} , \frac{(\vec{x}_{\alpha,\beta+1,\gamma}+\vec{x}_{\alpha,\beta-1,\gamma})\cdot\hat{e}_2}{2} , \frac{(\vec{x}_{\alpha,\beta,\gamma+1}+\vec{x}_{\alpha,\beta,\gamma-1})\cdot\hat{e}_3}{2}\right]^T$$

$$\mathbf{D}_w = \begin{bmatrix} (\vec{x}_{i+1,j,k} - \vec{x}_{i,j,k})\cdot\hat{e}_1 & 0 & 0 \\ 0 & (\vec{x}_{i,j+1,k} - \vec{x}_{i,j,k})\cdot\hat{e}_2 & 0 \\ 0 & 0 & (\vec{x}_{i,j,k+1} - \vec{x}_{i,j,k})\cdot\hat{e}_3 \end{bmatrix} \quad (61)$$

$$\mathbf{D}_{\alpha,\beta,\gamma} = \begin{bmatrix} \frac{(\vec{x}_{\alpha+1,\beta,\gamma}-\vec{x}_{\alpha-1,\beta,\gamma})\cdot\hat{e}_1}{2} & 0 & 0 \\ 0 & \frac{(\vec{x}_{\alpha,\beta+1,\gamma}-\vec{x}_{\alpha,\beta-1,\gamma})\cdot\hat{e}_2}{2} & 0 \\ 0 & 0 & \frac{(\vec{x}_{\alpha,\beta,\gamma+1}-\vec{x}_{\alpha,\beta,\gamma-1})\cdot\hat{e}_3}{2} \end{bmatrix}$$

Figure 27 shows a cell and cell nodes geometry. The definition of $\mathbf{D}_w$ effectively normalizes the $\vec{x}_w$ vector to have components between 0 and 1 for use in the weight functions $w_{\alpha,\beta,\gamma}$. The vector $\vec{x}$ gives the position of the evaluation point. The vector $\vec{x}_{\alpha,\beta,\gamma}$ is the location of the α, β, γ node, where α, β, γ are dummy indices for the $\hat{e}_1, \hat{e}_2, \hat{e}_3$ directions respectively. For each cell (e.g $\vec{x}_{\alpha,\beta,\gamma}$) node there exists a unique local node polynomial (e.g. $U_{\alpha,\beta,\gamma}$). Unlike the original Junkins method, the node polynomials are not centered at the node location (see Eq. 61) but instead at the midpoint of the neighboring 6 nodes. Each node polynomial has the argument $\vec{y}$ which is normalized by $\mathbf{D}_{\alpha,\beta,\gamma}$ to be valid from -1 to 1 in each direction. Each cell is part of the valid domain of 8 nodes. The choice of centering each node polynomial at the midpoint of its 6 neighboring nodes is taken to allow Chebhyshev measurement spacing during the least square polynomial fitting, as discussed later in the paper. The $w_{\alpha,\beta,\gamma}$ are Hermite weight functions[67] valid in the normalized (0, 1) space. The weight functions are always centered at the lower, left, front corner of each cell. The order of the weight functions can be chosen arbitrarily high to ensure any degree of user desired continuity or smoothness. The gradient of the potential is obtained by taking partial derivatives of the resulting polynomial, and higher order derivatives are also available

at a modest cost. Fitting only the potential function is memory efficient and leads to an *exact* formulation. In contrast, many previous models[67, 59, 28, 19] ignore exactness constraints, and fit directly the acceleration and any higher order derivatives[66] leading to significant memory growth.

It is worth re-emphasizing that the Junkins weighting function scheme[69] allows for arbitrary functions to be used for the local node interpolants. In the current study, simple polynomials are chosen to minimize runtime. The number of coefficients in each node polynomial is allowed to vary adaptively in each direction in order to maximize efficiency (defined as high accuracy and low memory).

The process of generating and using the Fetch model is implemented in two phases. Phase 1 is performed once off-line and consists of parallel computation of the local node polynomial coefficients on a global 3D grid. In order to deal with the singularity at the poles when working in the geocentric coordinates, a second rotated global grid is introduced. Details on the singularity problem and the rotated grid solution are given in the next section. Phase 2 is the runtime interpolation and involves a weighted evaluation of the local node polynomials. In practice, users of a Fetch model only interface with the runtime interpolation. The order of the weight functions (and accordingly the degree of continuity of the final composite function across the boundaries) is independent from the coefficient generation and can be selected at runtime. The next section gives a brief overview of the coefficient generation process.

3.5   Phase 1: Coefficient Generation

The Fetch interpolation model relies on a global discretization scheme for achieving adaptivity and continuity.[69] The whole solution space is divided into 3D cells in the spherical coordinate system consisting of polar angle (δ), azimuth (λ) and radial (r) directions. The geopotential within each cell is fit (in a least square sense) via node polynomials that are locally valid over the 8 cells touching that node. Hence, the

problem of fitting the global geopotential is reduced to finding localized polynomial coefficients corresponding to each cell node via least squares fitting.

3.5.1 Surface Discretization

Spherical coordinates are used for discretization of the whole solution domain outside the sphere given by the Earth's mean radius. The weighted interpolation scheme (used to achieve global continuity) requires a uniform discretization in each dimension. In other words, the cutting planes that partition the global domain into local cells must all be mutually orthogonal, although the spacing between the planes can vary. Hence the spacing of the cutting planes (specified by the diagonal components of $\mathbf{D}_w$ in Eq. 61) is a degree of freedom and can be utilized to improve efficiency. The polar angle, azimuth (δ, λ) space is ultimately chosen to have equal spacing for simplicity. Furthermore, the adaptive degree selection of the local polynomials affords the rationale that the smaller cells (in the Cartesian sense) near the poles will require polynomials of lower order to achieve the same accuracy as the larger cells near the equator. In addition, the two grid strategy used to remove the singularity also serves to keep a more uniform grid size in the δ, λ space when mapped to the surface. The cutting plane spacing in the radial direction is highly adaptive as the rapid undulations near the surface require shallow spacing while the high altitudes can be modeled with relatively distant spacings.

3.5.2 Singularity

A spherical coordinates based grid suffers from singularities at the poles when converting to and from the Cartesian coordinate space. A variety of methods are possible for dealing with the singularity, such as that used by Pines.[101] To avoid the burden of the extra dimension, we choose to remain with spherical coordinates, but use a rotated grid near the polar regions. Similarly, the Cubed-sphere model uses rotated mappings on the polar region "cube faces" to handle the singularity.[66] In the current

approach, the global grid is divided into two sub-grids, the primary grid and the rotated grid, both in the classic spherical coordinates. The primary grid suffers from singularities at the poles but its domain of validity is constrained to lie sufficiently far from both poles. Technically, according to Ref.[27] and from observation, the main grid is valid and numerically stable using spherical coordinates in all regions outside of a few degrees away from the poles. However, in this application, we enforce a more conservative domain, in order to avoid the small surface patches (and thereby save memory) that result from the uniform (δ, λ) grid. For maximum memory efficiency we find that a polar angle range of $x < \delta < (180 - x)$ is reasonable where x can be chosen anywhere from ~ 25 to ~ 50 degrees. Starting from the poles, a rotated grid is designed which has singularities at two points on the equator of the original grid (due to a 90 degree rotation about the x-axis). Hence, the rotated grid serves to remove the singularity at the poles in the original grid. Precise continuity between the two grid sub-models is maintained via a Hermite weighting function applied in a narrow overlapping band, as discussed in detail later.

Figure 28 shows the final surface discretization for the primary and the rotated grids. The active region for both the grids is shown as the darker shade, while the inactive region is lighter. The overlapping regions are indicated by the horizontal rings (and also bounded by the solid lines in Fig. 29). Evaluation inside the overlap region requires that both sub-models are computed and a 1D weighting function dependent only on the polar angle is used to ensure continuity across the sub-model boundaries.

Note that for the coefficient generation phase, each sub-model only requires coefficients for its respective domain including the overlapped areas. Therefore the two sub-model strategy creates almost no overhead during coefficient generation. Figure 29 shows a projection onto the Cartesian xy plane of the primary grid (o) overlaid with rotated grid (+) along with the overlapped region (bounded by the solid lines).
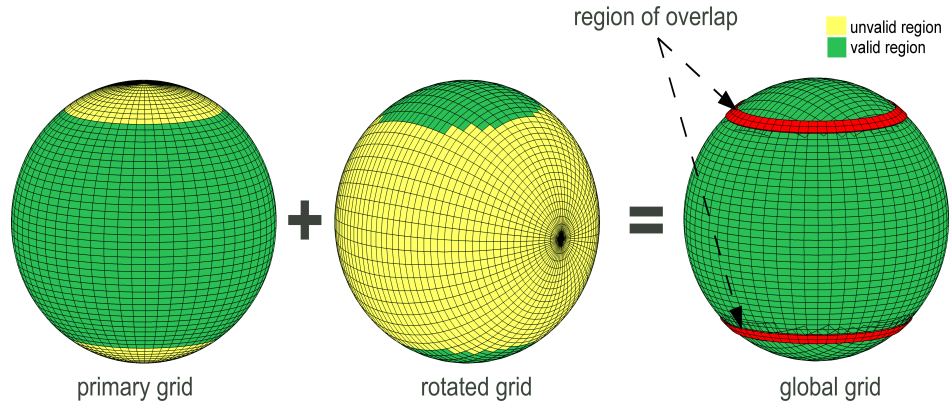
Figure 28: Surface discretization (overlap near 36 degree and 144 degree polar angle)
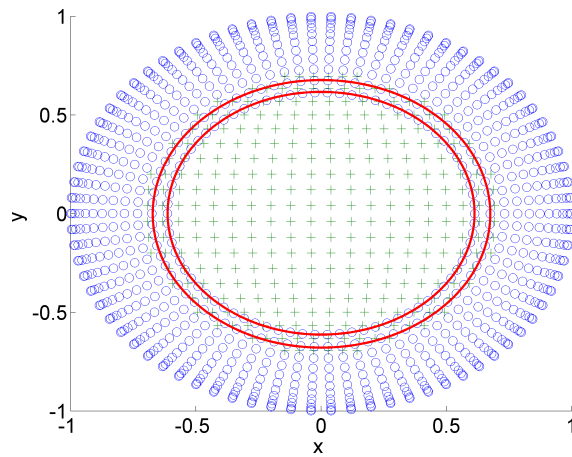
Figure 29: Overlapped region

3.5.3   Radial Discretization

Unevenly spaced shells are used to space the cutting planes in the radial direction extending all the way to the distance (approximately $230R_e$ for GGM03C model) beyond which the contribution from the SH terms higher than J_2 becomes less than machine double precision. The shells are densely packed near the surface where the field changes rapidly and they spread out as altitude increases. Having closely packed shells decreases the number of coefficients per cell but increases the number of cells required for the global model. On the other hand, choosing a large radial step size increases the number of coefficients per cell making runtime function evaluation slower. Hence, there is a trade-off between runtime memory and speed requirements.
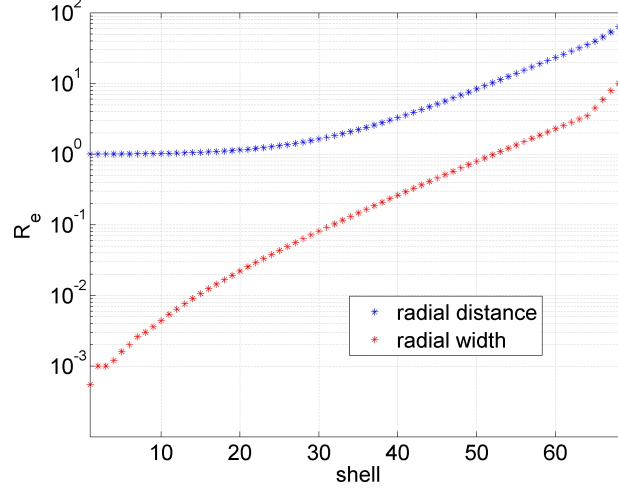
Figure 30: Radial discretization

Figure 30 shows the radial shell placement and width selected for the model used in this study, consisting of 68 shells. Starting from a log based distribution, manual tuning is performed to obtain the final shell spacing.

3.5.4 Adaptive polynomial selection

The Junkins weighted interpolation approach affords the freedom to choose any degree polynomial for the local approximation functions for a given node. This benefit is utilized in the polynomial fitting process by adaptively choosing a different degree polynomial for each cell node (see Fig 27).

$$Q_d = \sum_{a=0}^{d} \sum_{b=0}^{a} \sum_{c=0}^{a-b} C_\sigma y_1^{a-b-c} y_2^c y_3^b \qquad (62)$$

Equation 62 gives a general 3D polynomial considered during the fitting process. Vector $\vec{y}$ represents the normalized (-1 to 1) position of the evaluation point, $\vec{C}$ is the coefficients array, σ is the total cumulative loop counter and d stands for total degree of the polynomial. One approach to select the basis polynomial is to pick a value of d after a careful trade study, and then directly use the polynomial obtained from Eq. 62 over the whole solution domain. Junkins adopted this strategy during his early pioneering studies.[67,68] Given the computation power present at that time

and the intended application of his models, it was the most practical choice.

In our case, we want a broad range of candidate polynomials in order to finely tune the memory requirements of the resulting global model. Inserting $d=10$ into Eq. 62, results in a polynomial with $N=286$ coefficients. A set of candidate polynomials can be extracted from the general polynomial by selecting any non-repeating subset of the 286 terms. As an example, the number of possible ways to generate a set of 200 candidate polynomials (with no repeating coefficients) is $\frac{286!}{200!(286-200)!}$ or on the order of $\sim O(10^{74})$. To overcome this intractable problem, we extract a subset of polynomials by incrementally removing the highest order terms (total degree equal d) subject to a dependency on r. Removing such terms is justified because the radial grid spacing is variable, thus partially absorbing the need for adaptivity in those directions. Note that the summations in Eq. 62 are intentionally ordered such that the removed terms are always the trailing terms.

$$Q_2 = C_1 + C_2 y_1 + C_3 y_2 + C_4 y_3 + C_5 y_1^2 + C_6 y_1 y_2 + C_7 y_2^2 + C_8 y_1 y_3 + C_9 y_2 y_3 + C_{10} y_3^2$$

$$(63)$$

The algorithm to generate the candidate polynomial set starts by generating 9 major candidate polynomials (Q_d) from Eq. 62 for d ranging from 2 to 10. Next, we divide each of the 9 major polynomials into minor candidate polynomials $(Q_{d,z})$ by incrementally removing terms. Equation 63 shows the 1st major candidate polynomial for $d = 2$. It has 4 terms which are dependent on y_3 out of which 3 (the 3 of total degree 2) are removed in succession to generate 3 minor candidate polynomials (Eq. 64). We only need to delete the leading terms because the loop counter b changes slower than the loop counter for c (see Eq. 62).

$$Q_{2,3} = C_1 + C_2 y_1 + C_3 y_2 + C_4 y_3 + C_5 y_1^2 + C_6 y_1 y_2 + C_7 y_2^2 + C_8 y_1 y_3 + C_9 y_2 y_3$$

$$Q_{2,2} = C_1 + C_2 y_1 + C_3 y_2 + C_4 y_3 + C_5 y_1^2 + C_6 y_1 y_2 + C_7 y_2^2 + C_8 y_1 y_3 \qquad (64)$$

$$Q_{2,1} = C_1 + C_2 y_1 + C_3 y_2 + C_4 y_3 + C_5 y_1^2 + C_6 y_1 y_2 + C_7 y_2^2$$

Equation 64 shows the three minor polynomials $Q_{2,3}$, $Q_{2,2}$, $Q_{2,1}$ corresponding to the

major polynomial Q_2. The formula to calculate the number of terms to be removed (n_d) from a major polynomial of degree d is given by Eq. 65 . The total number of minor polynomials (219) equals to the sum of the number of terms removed for each major polynomial. Finally, the sum of 9 major and 219 minor polynomials together gives us a total set of 228 candidate polynomials $(P_{1...228})$. Algorithm 1 (see appendix A) outlines the complete candidate polynomial generation procedure.

$$n_1 = 1$$
$$n_i = n_{i-1} + i \quad \forall \; i = 2 \ldots d \tag{65}$$

During fitting, each of the candidate polynomials are ordered in terms of number of total coefficients. Starting with the candidate polynomial with the fewest coefficients (7 in current study), the least squares problem is evaluated. If a candidate polynomial solution is found with residuals acceptable according to user prescribed tolerances, then the polynomial is selected and the process is stopped for that node. Candidate polynomials with a greater number of coefficients are not evaluated. In this manner, the residuals of the local solutions will hover just below the user-supplied tolerance, leading to a uniform global residual distribution. In addition, because the candidate polynomials are evaluated in order of increasing memory footprint, each cell has a minimized memory requirement. Therefore the coefficients corresponding to each local node are chosen so as to minimize its memory footprint, subject to user provided residual constraints.

3.5.5 Localized least square approximation

The local approximation for each node polynomial is obtained via a least squares fit. As stated in the previous section, the minimum and maximum number coefficients allowed are fixed to 7 and 286 respectively. Given a candidate polynomial, the conventional least squares method for generating the coefficients is summarized by

Eq. 66.

$$\vec{C} = \mathbf{B}\vec{u} \tag{66}$$

Here, $\vec{C}$ is a $N \times 1$ vector of coefficient estimates, $\vec{u}$ is a $m^3 \times 1$ vector representing the measurements and $\mathbf{B}$ denotes the $N \times m^3$ least squares inverse matrix (Eq. 67):

$$\mathbf{B} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T \tag{67}$$

The matrix $\mathbf{H}$ represents the $m^3 \times N$ sensitivity matrix, m^3 represents the number of measurements (noting there are m measurements in each of the 3 directions) and N represents the total number of coefficients per node polynomial. The measurements are spaced according to the Chebyshev node distribution function given by Eq. 68 in each direction.[129] This choice of placements helps to minimize the well known Runge's phenomena and increases the robustness of the fit. As the radial shells are unevenly spaced, centering the node polynomials at the cell boundary (or cell nodes) destroys the Chebyshev measurement spacing and leads to non-uniform residual distributions over the node domain. Hence, to allow for Chebyshev spacing, the node polynomials are centered at the midpoints of the neighboring 6 nodes as measured from the current node. We note that our method differs from Junkins formulation as his polynomials are centered at the cell nodes which is a special case in our formulation when we choose constant cell sizes. Figure 31 shows the Chebyshev node distribution for $m = 11$ for a normalized 2D node domain.

$$x_i = \cos(\frac{2i - 1}{2m}\pi) \; , \; i = 1 \ldots m \tag{68}$$

The $\mathbf{H}$ matrix contains the first order partials of the candidate polynomial with respect to the estimated coefficients. The functional form for the rth row of $\mathbf{H}$ is given as follows:

$$H_{(r,:)} = \frac{\partial P_i}{\partial \vec{C}}\bigg|_{\vec{\rho}_r} \tag{69}$$
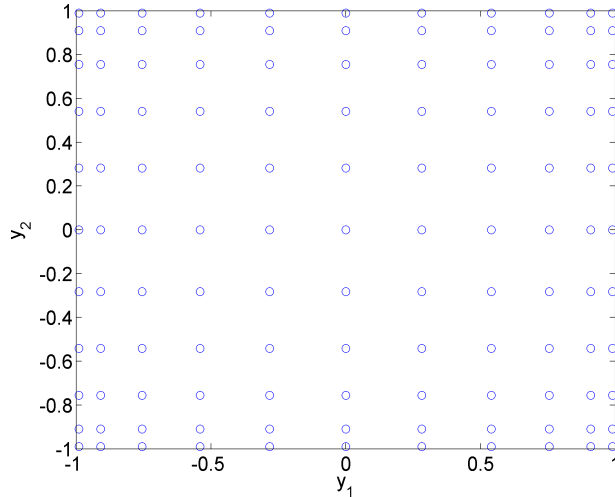
Figure 31: 2D measurement distribution

where $\vec{\rho}_r$ is the measurement location, and P_i is the ith candidate polynomial. Computing the analytic partials and evaluating them at the specific measurement locations, the full H matrix for each candidate polynomial is computed using a symbolic manipulator program (Maple) and results in a $m^3 \times N$ matrix of rational fraction entries.

Each node polynomial is normalized between -1 to 1 along each of its dimensions in its node domain. If the number of measurements (m^3) and their respective positions (defined by the Chebyshev spacing) do not change, then the fully evaluated **H** matrix is invariant across all nodes for a given candidate polynomial. This static property of the **H** matrix (as pointed out by Junkins[67] and Lekien and Marsden[81]) is only true if the number of measurements and their relative positions ($\vec{\rho}_r$ for all measurements) are the same across all node domains. Given that the **H** matrix is static, the inversion matrix that solves the least squares problem in Eq. 67 can be computed analytically just once. In other words, once the **B** matrix is solved analytically, there is no need to numerically solve a linear system in order to obtain the least squares coefficient fit for a given candidate function. Therefore, in order to test a single candidate polynomial across the global domain, each cell simply requires one matrix

66

multiplication (Eq. 66). It is further emphasized that the $\mathbf{H}$ matrix, when evaluated with a symbolic manipulator is composed of rational fractions. Accordingly, $\mathbf{B}$ from Eq. 67 can be analytically computed and the results are also in the form of rational fractions. Therefore, the typical numerical problems associated with solving large linear systems is completely avoided. Quoting from[68] the *"one inverse property can lead to order of magnitude savings in computer time for large data sets"*.

As an example, the matrix $\mathbf{B}$ for the candidate polynomial having 7 coefficients (N=7) and with m=11, is dense with explicit cosine functions present in most of its terms (owing to the Chebyshev spacing from Eq. 68) . Equation 70 gives the first row of this matrix. The actual size of the matrix is 7×11^3.

$$
\mathbf{B(1,:)} = \left[\begin{array}{c}
\frac{7}{14641} - \frac{36}{14641} \cos\left(1/11\,\pi\right) + \frac{8}{14641} \cos\left(3/11\,\pi\right) + \frac{8}{14641} \cos\left(\frac{5}{11}\,\pi\right) - \frac{8}{14641} \cos\left(4/11\,\pi\right) - \frac{8}{14641} \cos\left(2/11\,\pi\right) \\
-\frac{2}{1331} \cos\left(1/22\,\pi\right) \\
-\frac{2}{1331} \cos\left(1/22\,\pi\right) \\
-\frac{2}{1331} \cos\left(1/22\,\pi\right) \\
\frac{4}{14641} + \frac{36}{14641} \cos\left(1/11\,\pi\right) - \frac{8}{14641} \cos\left(3/11\,\pi\right) - \frac{8}{14641} \cos\left(\frac{5}{11}\,\pi\right) + \frac{8}{14641} \cos\left(4/11\,\pi\right) + \frac{8}{14641} \cos\left(2/11\,\pi\right) \\
\frac{2}{1331} \cos\left(1/11\,\pi\right) + \frac{2}{1331} \\
\frac{4}{14641} + \frac{36}{14641} \cos\left(1/11\,\pi\right) - \frac{8}{14641} \cos\left(3/11\,\pi\right) - \frac{8}{14641} \cos\left(\frac{5}{11}\,\pi\right) + \frac{8}{14641} \cos\left(4/11\,\pi\right) + \frac{8}{14641} \cos\left(2/11\,\pi\right)
\end{array} \right]^T
\tag{70}
$$

For this study a Maple worksheet is used to analytically invert the $\mathbf{B}$ matrices corresponding to all possible candidate polynomials. The worksheet takes the maximum polynomial degree O_{max} and m as inputs and writes the $\mathbf{B}$ matrices to a file for later use by the coefficient generation routines. Analytic inversion of the normal matrix $H^T H$ of dimension $m^3 \times m^3$ is computationally intensive. For m on the order of 10, such inversions were simply not possible on a typical desktop computer until recently. However it is emphasized that these inversion matrices only need to be computed just once for a given interpolant and a fixed measurement spacing scheme. The number of observations in each direction is kept at a minimum of $O_{max}+1$ in order to preserve a well posed (over constrained) least squares problem. Once the measurements are obtained and the measurement vector is formed, only a single matrix multiply is needed

to obtain the least squares coefficients. Exact and precomputed $\mathbf{B}$ matrices ensure a fast, accurate, and numerically well-conditioned coefficient generation process.
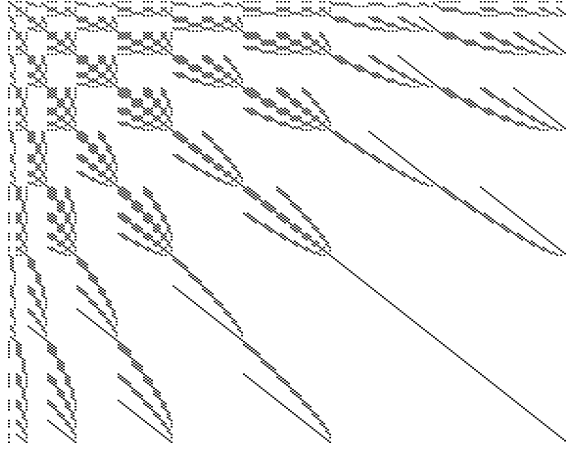
Figure 32: Structure of $(\mathbf{H}^T\mathbf{H})^{-1}$ for m=11 and N=286; black dots = non-zero terms

For the current study, the value of m is fixed at 11 for all the node domains and the maximum polynomial degree (O_{max}) is fixed at 10. Figure 32 gives the structure of $(\mathbf{H}^T\mathbf{H})^{-1}$ matrix for m=11 and N=286. Here, the black dots represent the non-zero terms in the matrix. The size of this matrix is 286×286. Maple version 13 was used for this study and it took approximately 36 hrs on a single workstation (see Table 13) to generate the $\mathbf{B}$ matrices for all 228 candidate polynomials.

3.5.6 Scalable SH degree selection

Once all the $\mathbf{B}$ matrices are generated, the measurement $\vec{u}$ for each node domain must be computed in order to obtain the coefficients for all the candidate functions using Eq. 66 . The measurement vector requires evaluation of the SH function at each of the m^3 locations within each node domain. Noting there are $\sim$8 million node domains for the global 360×360 resolution case, obtaining the measurement vector for m=11 requires evaluation of the SH function on the order of 10 billion times. Fortunately the full precision SH field is not necessary at all altitudes as the high order terms become undetectable to machine precision as the altitude increases.

Accordingly, a scalable SH degree selection method is adopted. Figure 33 shows the altitudes where the contributions of higher order terms of the SH expansion (given by the GGM03C model) become less than a normalized 1E-15 (near machine precision for double). For example at a radial distance of $10R_e$ (equivalently at an altitude of $9R_e$), only terms up to degree and order 30 are necessary for the computation of the measurement vector. Using 1E-15 as a cut-off tolerance is overly conservative considering that the max normalized residuals in the target SH field fits (to be discussed later) are always substantially larger.
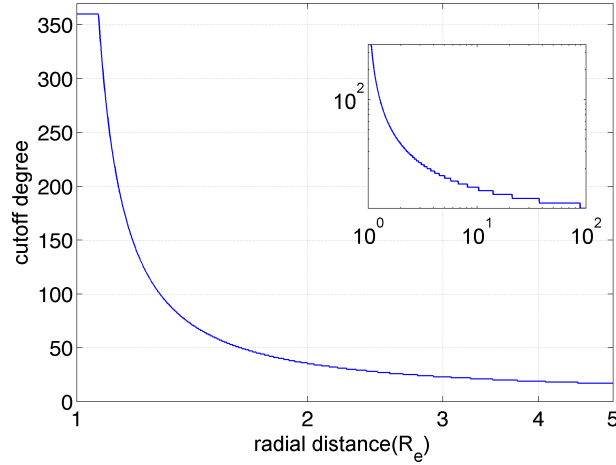
Figure 33: Degree and order selection curve

The equation used to generate the curve in Fig. 33 is given in Eq. 71 and is found via a non linear least squares curve fit of a large sampling of measurements taken across the global domain. Equation 71 takes in normalized radial distance (r) as input and returns the degree and order value (F_{siz}) which is the max degree and order SH gravity field necessary for that radial distance.

$$F_{siz} = \min \left[360, floor(\frac{20.37185996049265}{\ln(r)^{1.165605077780336} + 1.625250344642757E - 2} + 4.461932685300188) + 1 \right] \tag{71}$$

It is evident from Fig. 33 that for most of the global domain the high order spherical harmonics terms are not significant even in double precision arithmetic. As the evaluations move radially outwards, the measurement vector computation time decreases

69

rapidly thereby significantly speeding up the coefficient generation process. Note that Eq. 71 can be used to speed up the computation of SH for general applications as well.

3.5.7 Continuity

The Fetch model achieves continuity in any order by utilizing the weighted interpolation scheme (developed by Junkins[69, 67]) which leads to eight interpolant function evaluations instead of one for each composite function call. The eight interpolants applicable to each cell are evaluated in their own -1 to 1 normalized domain and the weighting is done in the overlapping space inside a cell, normalized from 0 to 1. The normalization allows the use of Hermite weighting functions which enable continuous higher order derivatives. The weight function as they appear in Eq. 61 can be found in Eq. 5a-5g of Ref.[67] Equation 72 summarizes the final Fetch approximation for the potential and first derivative with respect to x_1, x_2 and x_3, which are the spherical components of position vector defining the evaluation point. $\vec{C}$ stands for the coefficients of the node polynomial under consideration, $w_{i,j,k}$ are the Hermite weight functions at the i, j, k node as specified by Junkins, and $N_{i,j,k}$ is the number of coefficients for the i, j, k node polynomial. These partial derivatives are further converted into final Cartesian acceleration by applying the inverse of the classic spherical coordinate transformation. Higher order derivatives are obtained by taking further

derivatives of Eq. 72 and applying the chain rule.

$$U_F = \sum_{\alpha=i}^{i+1} \sum_{\beta=j}^{j+1} \sum_{\gamma=k}^{k+1} w_{\alpha,\beta,\gamma}(\vec{x}_w) U_{\alpha,\beta,\gamma}(\vec{y})$$

$$\frac{\partial U_F}{\partial x_1} = \sum_{\alpha=i}^{i+1} \sum_{\beta=j}^{j+1} \sum_{\gamma=k}^{k+1} \left[\frac{\partial w_{\alpha,\beta,\gamma}(\vec{x}_w)}{\partial \vec{x}_w} \frac{\partial \vec{x}_w}{\partial x_1} U_{\alpha,\beta,\gamma}(\vec{y}) + w_{\alpha,\beta,\gamma}(\vec{x}_w) \frac{\partial U_{\alpha,\beta,\gamma}(\vec{y})}{\partial \vec{y}} \frac{\partial \vec{y}}{\partial x_1} \right]$$

$$\frac{\partial U_F}{\partial x_2} = \sum_{\alpha=i}^{i+1} \sum_{\beta=j}^{j+1} \sum_{\gamma=k}^{k+1} \left[\frac{\partial w_{\alpha,\beta,\gamma}(\vec{x}_w)}{\partial \vec{x}_w} \frac{\partial \vec{x}_w}{\partial x_2} U_{\alpha,\beta,\gamma}(\vec{y}) + w_{\alpha,\beta,\gamma}(\vec{x}_w) \frac{\partial U_{\alpha,\beta,\gamma}(\vec{y})}{\partial \vec{y}} \frac{\partial \vec{y}}{\partial x_2} \right]$$

$$\frac{\partial U_F}{\partial x_3} = \sum_{\alpha=i}^{i+1} \sum_{\beta=j}^{j+1} \sum_{\gamma=k}^{k+1} \left[\frac{\partial w_{\alpha,\beta,\gamma}(\vec{x}_w)}{\partial \vec{x}_w} \frac{\partial \vec{x}_w}{\partial x_3} U_{\alpha,\beta,\gamma}(\vec{y}) + w_{\alpha,\beta,\gamma}(\vec{x}_w) \frac{\partial U_{\alpha,\beta,\gamma}(\vec{y})}{\partial \vec{y}} \frac{\partial \vec{y}}{\partial x_3} \right]$$

$$U_{\alpha,\beta,\gamma}(\vec{y}) = \sum_{a=0}^{O_{max}} \sum_{b=0}^{a} \sum_{c=0}^{a-b} C_\sigma y_1^{a-b-c} y_2^c y_3^b \quad \forall\, \sigma \leq N_{\alpha,\beta,\gamma}$$

$$\frac{\partial U_{\alpha,\beta,\gamma}(\vec{y})}{\partial y_1} = \sum_{a=0}^{O_{max}} \sum_{b=0}^{a} \sum_{c=0}^{a-b} C_\sigma (a-b-c) y_1^{a-b-c-1} y_2^c y_3^b \quad \forall\, \sigma \leq N_{\alpha,\beta,\gamma},\; a-b-c > 0$$

$$\frac{\partial U_{\alpha,\beta,\gamma}(\vec{y})}{\partial y_2} = \sum_{a=0}^{O_{max}} \sum_{b=0}^{a} \sum_{c=1}^{a-b} C_\sigma c\, y_1^{a-b-c} y_2^{c-1} y_3^b \quad \forall\, \sigma \leq N_{\alpha,\beta,\gamma}$$

$$\frac{\partial U_{\alpha,\beta,\gamma}(\vec{y})}{\partial y_3} = \sum_{a=0}^{O_{max}} \sum_{b=1}^{a} \sum_{c=0}^{a-b} C_\sigma b\, y_1^{a-b-c} y_2^c y_3^{b-1} \quad \forall\, \sigma \leq N_{\alpha,\beta,\gamma}$$

(72)
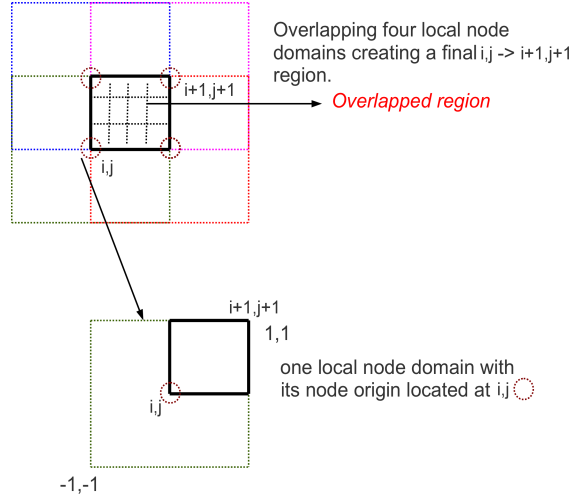
Figure 34: Continuity in 2D from weighted evaluation

Figure 34 summarizes the weighting function technique for two equally spaced dimensions. Equation 73 shows the final weighted evaluation of the potential function in the small region of sub-grid overlap (see Fig. 29).

$$U_\Omega = U_{pri}(\delta, \lambda, r)\Omega(\phi) + [1 - \Omega(\delta)]U_{rot}(\delta, \lambda, r)$$ (73)

In Eq. 73 δ refers to normalized polar angle which varies between 0 and 1 across the overlapped region. U_{pri} and U_{rot} are the interpolated geopotentials evaluated on the

primary and the rotated grids, respectively. U_Ω is the final weighted geopotential in the overlapping region, and Ω represents a 7th degree 1D Hermite weight function which achieves 3rd order continuity (determined by the outer term δ^4) and is given by Eq. 74.

$$\Omega(\delta) = \delta^4(35 - 84\delta + 70\delta^2 - 20\delta^3) \tag{74}$$

Equations 73 and 74 can be further differentiated with respect to δ using the chain rule and by using Eq. 61 to obtain continuous derivatives up to the desired order in the overlap region. Hence, due to the weighting evaluation and the overlapping techniques, continuity and exact higher order derivatives of the interpolated geopotential are maintained globally. Exact higher order derivatives possess attractive dynamical properties especially for applications such as orbit determination and trajectory optimization. Figure 36 illustrates the continuity by showing the potential and its first three derivatives with respect to Cartesian x for the sub-grid overlapping region. Figure 36 (left) illustrates continuity in all four cases, achieved by choosing a 7th degree Hermite weighting function which is continuous upto 3rd order (smooth upto 2nd order). Kinks and discontinuities in the higher order derivatives are illustrated in Fig. 36 and result due to the use of a lower order (Fig. 35) weight function.

3.5.8 Residual tolerances

The final acceptable residual level (when compared to SH fitting function) for each node domain is a function of four sub-tolerances: the RMS and max of the potential residual and the RMS and max of the acceleration residuals (norm of the difference of the Cartesian acceleration vectors). The potential residual tolerance is adaptively determined based on the radial distance of the current point in the node domain and the degree and order of the SH function being fit. The target value is chosen to conservatively mirror the expected errors of the SH function. Estimated accuracies of the GGM03C solution are given in Ref.[132] and the associated release notes. The
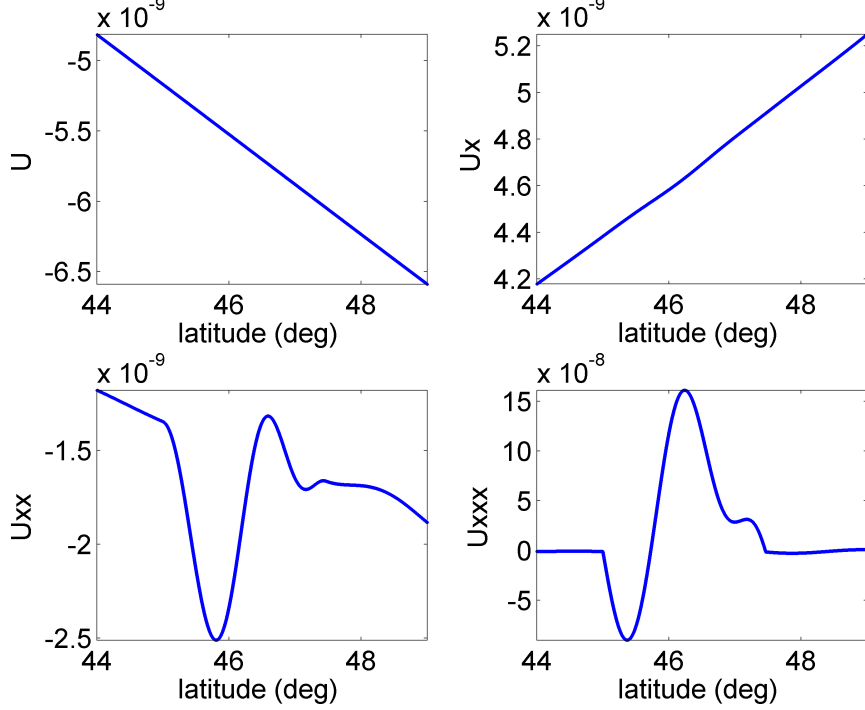
Figure 35: Weight functions continuous to 3rd order : 70×70 field

accumulated error as a function of SH degree is replicated in Fig. 37. For example, up to degree and order 70, the accumulated error for the geoid height is ~ 6 mm or $\sim$ 1E-9 in normalized units. Up to degree and order 360, the accumulated normalized error is $\sim$ 3E-8. Therefore the confidence of the potential evaluation at the surface of a 360×360 and 70×70 field is approximately 8 and 9 digits of accuracy respectively. The accumulated error curve in Fig 37 serves as a baseline target for the interpolation residuals for geopotential evaluation at the surface. In order to map the target residuals to different altitudes, Eq. 75 includes the (R_e/r) term similar to the structure of the interpolant. For a given size of the SH field, the covariance matrix terms (σ_c and σ_s) from the GGM03C solution are obtained.[132] To be conservative the baseline target potential residual (τ_U) is always kept below a user defined constant η_0, which is equal to 5E-9 for the current study.

$$\tau_U(r) = \min \left(\eta_0 \ , \ \sqrt{\left[\sum_{i=0}^{d} (\frac{R_e}{r})^{2i} \sum_{j=0}^{i} (\sigma_c(i,j)^2 + \sigma_s(i,j)^2) \right]} \right) \qquad (75)$$
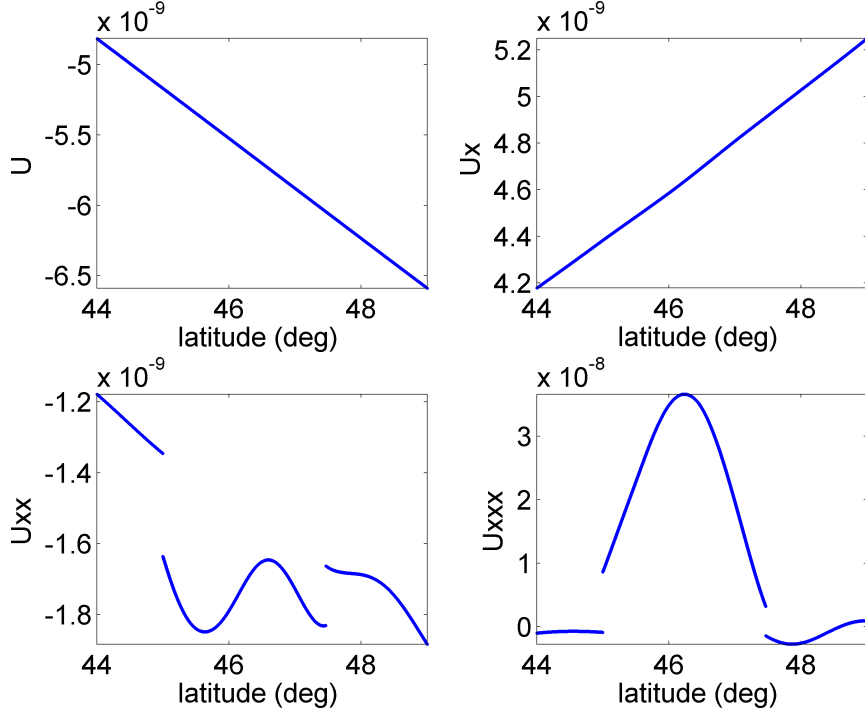
Figure 36: Weight functions continuous to 1st order (right) : 70×70 field

For a given degree and order of the SH field (d), a residual scaling graph is obtained using Eq. 75. Figure 38 illustrates the curve for various degree and order truncations. The R_e/r term dominates at the high altitudes, making the baseline target residuals approximately the same for all four cases shown. This adaptive scaling in the radial direction provides a context for targeting interpolation residuals, allowing for a highly optimized memory footprint of the global model. The baseline target value of τ_U multiplied by a user defined constant (η_1) serves as the final potential RMS residual tolerance used during the fitting process. The acceleration RMS residual tolerance is directly obtained by multiplying the potential RMS residual tolerance by a user defined constant (η_2). The max residual tolerances on both the potential and acceleration are obtained by multiplying the corresponding per node domain RMS values (computed during the fitting process), by user defined constants, η_3 and η_4, respectively. Equation 76 gives the four sub-tolerances that are required to be met in order for a candidate polynomial to be selected. Here, $\Upsilon_{1...4}$ represent the final
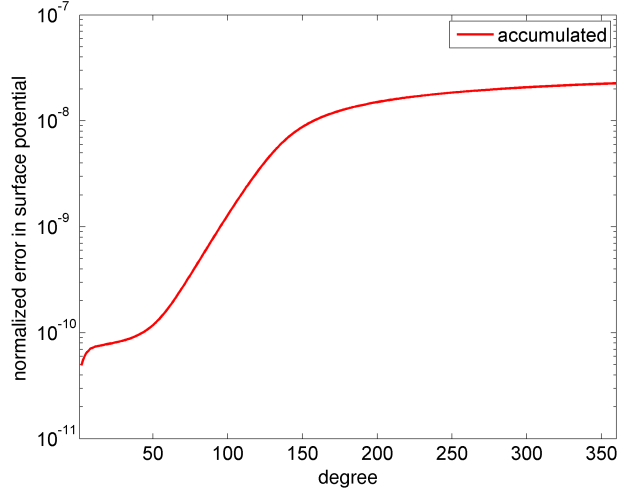
Figure 37: GGM03C accumulated surface error

potential RMS, acceleration RMS, potential max and the acceleration max residual tolerances respectively. Its should be noted that κ_U and $\kappa_{\vec{\nabla}U}$ denote the per node domain RMS values for the potential and acceleration, respectively.

$$
\begin{aligned}
\Upsilon_1 &= \tau_U \ \eta_1 \\
\Upsilon_2 &= \Upsilon_1 \ \eta_2 \\
\Upsilon_3 &= \kappa_U \ \eta_3 \\
\Upsilon_4 &= \kappa_{\vec{\nabla}U} \ \eta_4
\end{aligned}
\tag{76}
$$

It can be argued, because the residuals of the interpolation model are within the published accuracy of the SH model, that neither the fitted SH model nor the interpolation model is more likely to represent the true geopotential. Despite such an argument, the additional constraint to limit residuals in the accelerations is included in order to maintain consistency in a uniform manner for the acceleration results from both the GGM03C SH and the Fetch models. No constraints are placed on higher order derivatives because there is no justification for exact consistency. To the contrary, applications such as orbit determination and trajectory optimization require exact derivatives of the function being used and not the function being approximated.

75
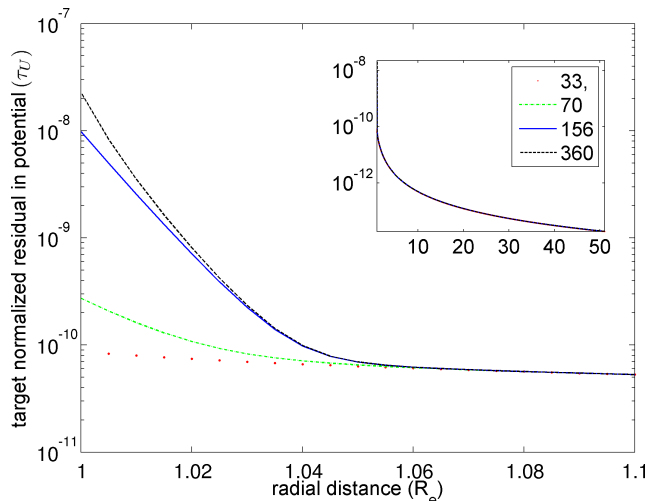
Figure 38: Residual scaling graph (using $\eta_0=1$)

3.5.9   Parallel coefficient generation

In spite of various algorithmic optimizations as stated in the previous sections, the coefficient generation process for a complete 360×360 global model would require on the order of 12,000 CPU hours using a modern workstation (2.27 GHz Intel Xeon E5520 processor). Note that the 360×360 degree and order model contains approximately 8 million nodes and up to 228 candidate functions are evaluated for each node, where each candidate function contains up to 286 coefficients. Furthermore, each node includes $11^3=1{,}331$ measurements, and each SH call can include up to $\sim 130{,}000$ terms. As the coefficients need to be generated just once, the process is parallelized using the MPI programming model. The main computation burden comes from SH field computation and increases near the surface where higher order terms are significant. Depending on the radial distance and the local geopotential characteristics of the node under consideration, the fitting times can vary significantly, leading to non-homogeneity in computation over the global domain.

To tackle the non-homogeneity, a parallel algorithm using a master-worker strategy is implemented for computing the coefficients. A general version of the algorithm is given in Appendix A (algorithm 3). The first CPU thread is made the master

thread and is responsible for gathering and distributing work (nodes to be fit) to all other CPUs or worker threads. The worker thread comes back to the master thread after fitting its share of allocated nodes and waits for more nodes. Each worker thread writes its own separate coefficient file and the master thread is responsible for joining all the files into the final coefficient binary file. The master-worker algorithm leads to a 1.37 fold speedup over the case of static assignments for each thread. The algorithm also has attractive features like load balancing and auto resume which are useful if computation is interrupted for any reason. The parallel version of the code is implemented in Fortran and can be compiled using either the Intel MPI compiler or the OPEN-MPI compiler. The complete process, as described, for fitting a 360 degree and order field across the full domain takes 11 hrs. on a cluster of 1,100 processors (3.33 GHz Intel Xeon X5680 processor). For the current study, the TACC Lonestar Linux Cluster is used consisting of 1,888 compute nodes with 6 cores per node, resulting in a total of 22,656 cores and a peak compute performance of 302 TFLOPS.
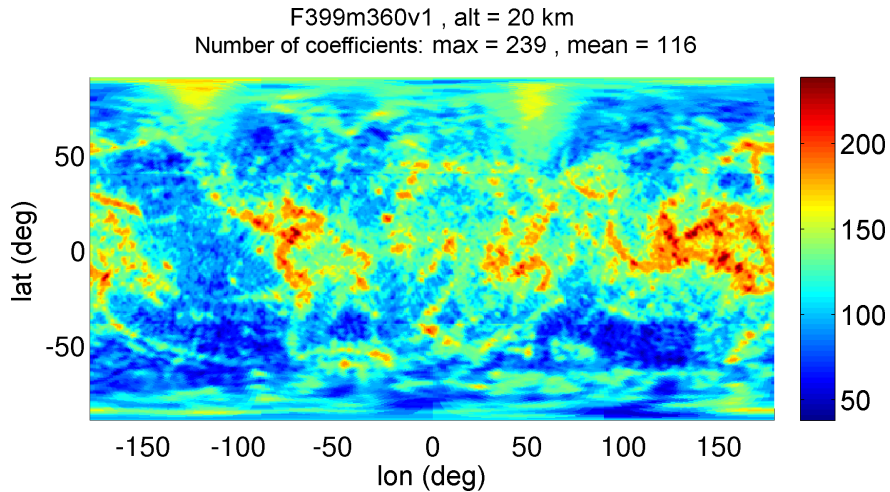
Figure 39: Distribution of number of coefficients at 20 km altitude : 360×360 field

Figures 39 to 42 shows a surface distribution for number of polynomial coefficients of the optimally selected polynomials at altitudes ranging from 20 to 20,000 km for the 360×360 model. At all altitudes, the (δ, λ) grid size remains the same; hence, we observe a reduction in number of coefficients as altitude is increased. The regions

F399m360v1 , alt = 200 km
Number of coefficients: max = 151 , mean = 63.1
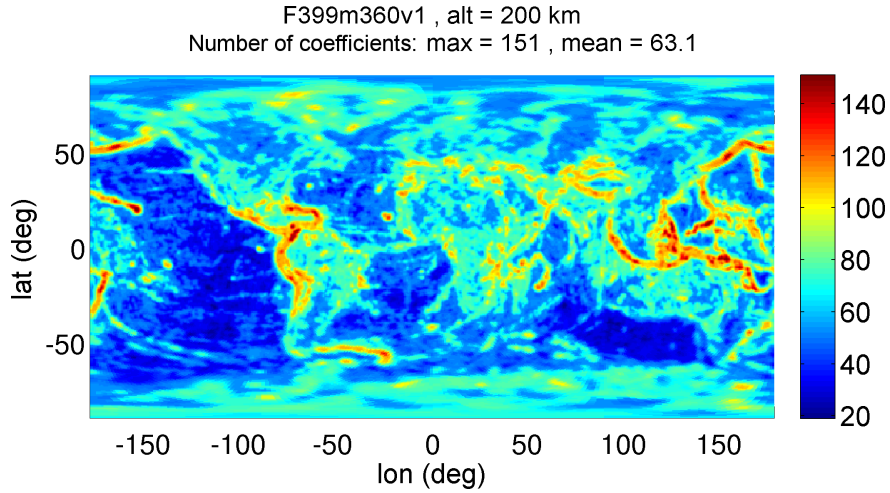
Figure 40: Distribution of number of coefficients at 200 km altitude : 360×360 field

F399m360v1 , alt = 2000 km
Number of coefficients: max = 45 , mean = 34.6
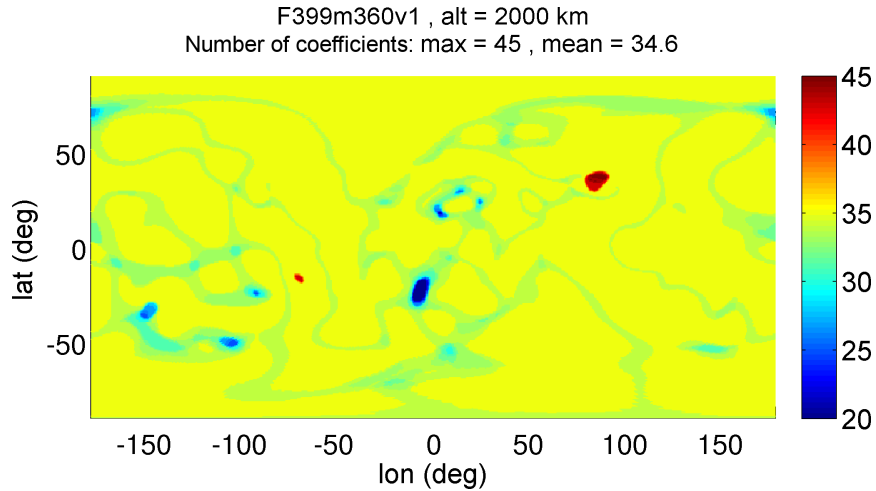
Figure 41: Distribution of number of coefficients at 2,000 km altitude : 360×360 field

with higher numbers of coefficients correspond to the regions where the geopotential changes rapidly. Figure 43 shows the contour of the radial acceleration (in mGals with two body and J_2 terms removed) evaluated at the surface for 360×360 field. A bigger version of Fig. 43 is given in Appendix B. The computed coefficients for the primary and rotated grids, along with the required meta-data, are stored in one dense binary file.
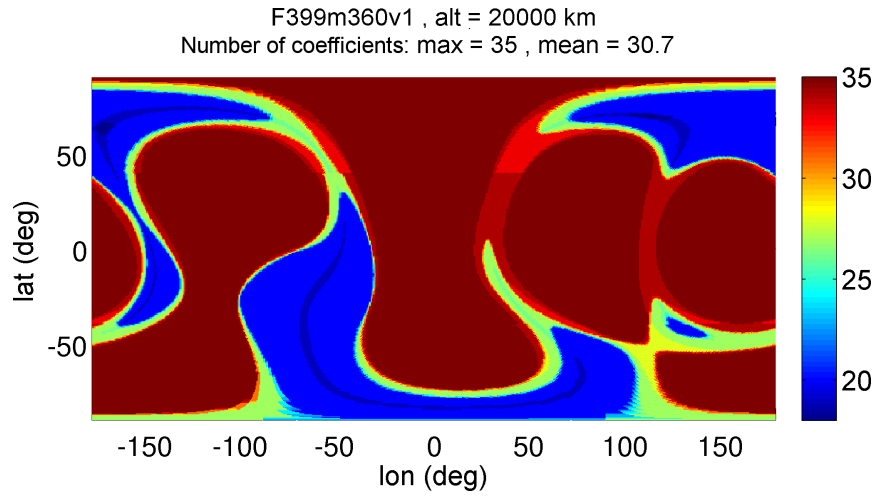
Figure 42: Distribution of number of coefficients at 20,000 km altitude : 360×360 field
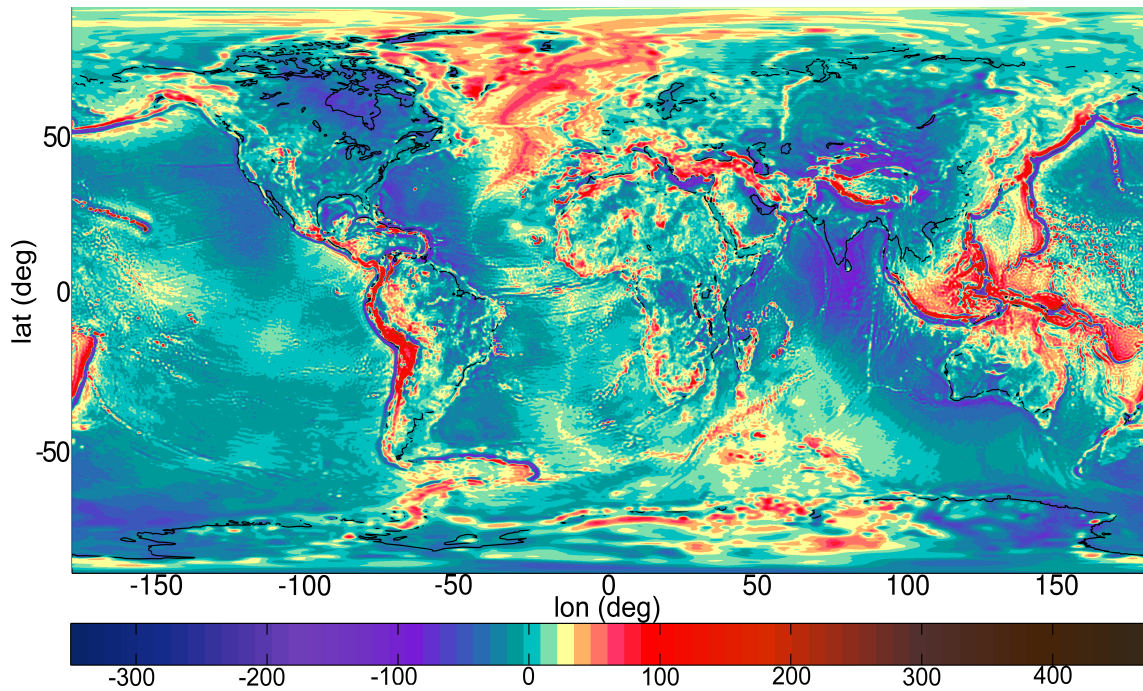
Figure 43: Radial acceleration (in mGals) for 360×360 GGM03C field at surface (two body and J_2 terms removed)

Table 11: Fetch models

Model name	SH field	η_1	η_2	η_3	η_4	Γ	Memory	# Cells	# Coefficients	Expected[1] speedup
F399m33v1	33×33	1.00	10.00	11.00	10.00	2.00	121.44 MB	175,536	15,828,234	8.5
F399m70v1	70×70	1.00	10.00	11.00	11.00	1.53	359.91 MB	564,075	46,891,426	28.7
F399m156v1	156×156	1.00	12.00	11.00	11.00	1.21	897.57 MB	2,015,076	116,628,043	206.6
F399m360v1	360×360	1.00	20.00	11.00	8.00	0.92	2.36 GB	7,615,254	312,545,677	2914.2

3.5.10   Model classification

Given the SH base field, the Fetch model is generated by specifying the four residual constants (defined previously, $\eta_{1...4}$) and a grid resolution control parameter called Γ.

$$s = \Gamma \sqrt{\frac{180}{d}} \tag{77}$$

In Eq. 77 s is the cell edge length in degrees, and d is the degree and order of the SH base field. Γ provides a dial to control the memory vs. speedup trade. The value of Γ directly affects the polynomial fitting accuracy and Fetch runtime performance. A high value results in a final Fetch model with a smaller memory footprint but slower runtime performance and vice versa. Typical values for Γ lie between 0.5 to 2 for various SH base fields considered in this study. Table 11 lists the various Fetch models generated for this study and are classified based on the degree and order of the SH field that is fit, residual tolerances ($\eta_{1...4}$) and Γ. The naming convention includes the SPICE body number,[2] the SH model degree and order, and the version number of the current release. The F, m, and v stand for Fetch, model, and version, respectively. The memory footprint increases almost linearly with increase in the SH degree and order.

Figures 44 and 45 shows various Fetch models with their memory footprint and expected speedup as a function of Γ. Two classes of Fetch models corresponding to SH field sizes 33×33 and 70×70 are shown. Increase in both, the speedup and memory is observed with a decreasing value of Γ. For fairness, the residual tolerances ($\Upsilon_{1...4}$) within each class of models are kept constant. The residual tolerances used in Fig. 45 for the 70×70 models are more relaxed (by a factor of 1.2 on η_2 and η_4) than for the models mentioned in Table 11. Only the models satisfying the residual
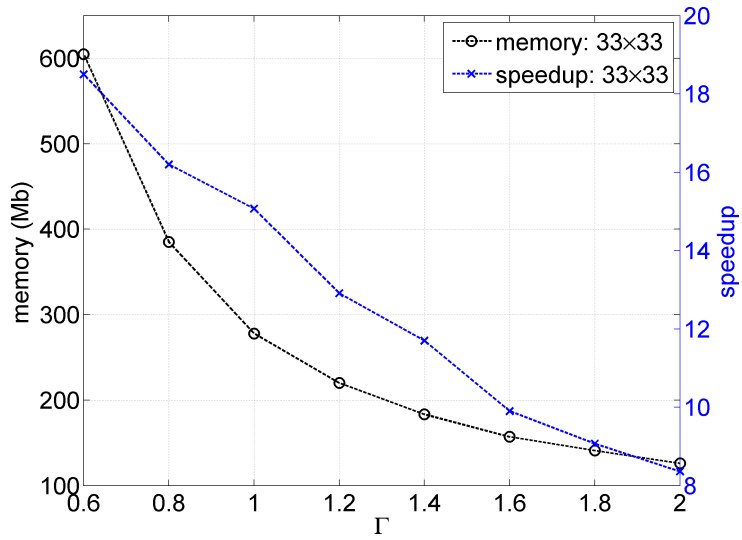
Figure 44: Memory and Speedup vs Γ : 33×33 Fetch model
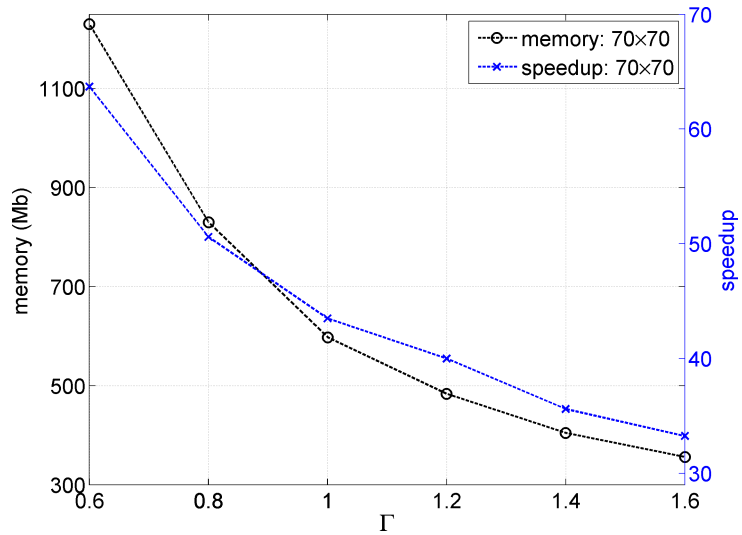
Figure 45: Memory and Speedup vs Γ : 70×70 Fetch model

tolerances are plotted, thereby imposing an upper limit on Γ.

3.6   Phase 2: Runtime Evaluation

The runtime implementation of a Fetch model is designed to be as simple as possible. The steps required at runtime are 1. initialize Fetch (e.g. load the binary coefficient file), and 2. call Fetch runtime routines to retrieve the potential and any desired derivatives.

The first step initializes Fetch and loads the coefficient file requested by the user to the computer's Random Access Memory (RAM). This step is executed only once during the model initialization phase. This step can take anywhere from less than a second to up to ~20 seconds depending on the size of the Fetch model being loaded and hardware of the computer. Once the model is initialized, various calls to Fetch routines can proceed. Inside the core runtime routines, the coefficients are efficiently tracked and the correct 8 neighboring node polynomials are identified and evaluated followed by a final weighted evaluation of the composite Fetch runtime function.

3.6.1 Coefficient lookup

All routines take the spacecraft geocentric Cartesian position vector as part of their input and subsequently identify the 3D cell housing that position vector. Normally, such a task would require a lookup table, however the uniform surface grid spacing along with the precomputed radial shell distances can be exploited to identify the correct cell using only a double to integer conversion function. The coefficients are stored in manner such that only one node position lookup is required in order to gather all of the neighboring eight node positions.

3.6.2 Fetch runtime routines

Table 12 lists the runtime routines that have been implemented for the current release. The main "get_Fetch" routine takes the geocentric Cartesian vector, body GM, body radius, J_2 and required derivative order as inputs and gives the interpolated potential plus derivatives as output. The accelerations are computed by taking the gradient

Table 12: Available Fetch routines

Routine name	Task performed
init_Fetch	Initializes Fetch + coefficient file
get_Fetch	Computes potential + derivatives

Table 13: Test hardware/software

Component type	Component
CPU	Intel Core i7 950 @ 3.07 Ghz
RAM (Memory)	6.0 GB
Compiler	Intel Fortran 12.0
Operating System	Ubuntu 12.10

of the interpolated geopotential function with respect to spherical coordinates, then

performing the necessary transformations to the Cartesian coordinates.[127] Similar coordinate transformations and chain rules are required for any necessary higher order derivatives. In this implementation, runtime routines are provided to include up to third order derivatives (with respect to Cartesian coordinates) of the potential. Again, it is emphasized that the singularity issue with the spherical coordinates conversions is handled through the use of the two overlapping grids.

3.7   Runtime Performance

In this section the performance of the Fetch models are evaluated against an optimized CPU implementation of the Pines SH model.[101,85] It is noted that both the Pines SH algorithm and the Fetch model are singularity free. Even though the Fetch model extends beyond the Moon, the performance comparisons are limited to an altitude of approximately $5R_e$. All four of the Fetch models from Table 11 are considered for performance comparison. Table 13 gives the specifications of the runtime test hardware and compiler.

3.7.1   Comparison with fitting SH function

Comparisons are performed by dividing the evaluation region into five altitude bands listed in Table 14 . For each band, sample direct calls to Pines SH and Fetch models are made and differences between the Fetch and SH models are evaluated. The number of sample points for each band vary between 6,500 and 50,000 and are selected heuristically depending on the degree and order of the field.

Table 14: Performance evaluation regions

Band id	Start altitude (km)	End altitude (km)
Band 1	36.0	65.0
Band 2	65.0	1,000.0
Band 3	1,000.0	2,550.0
Band 4	2,550.0	6,378.0
Band 5	6,378.0	19,135.0
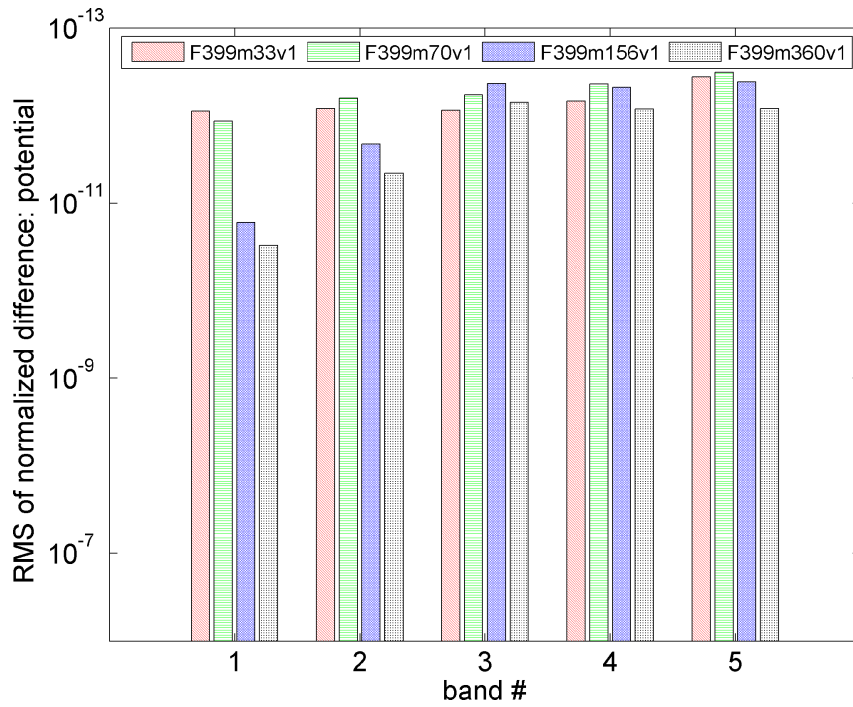
Figure 46: Normalized difference (RMS) in potential when compared to SH
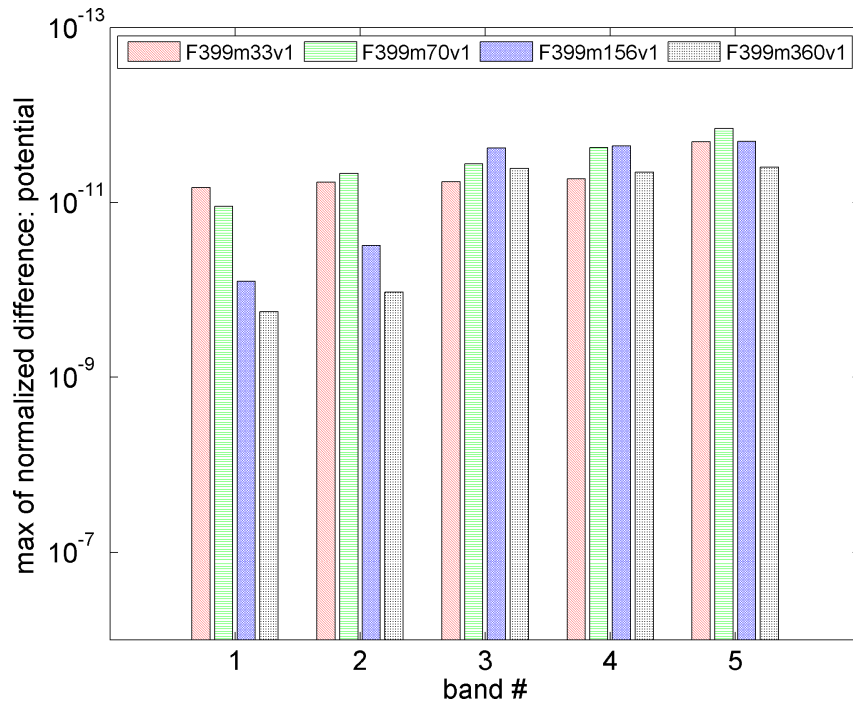
Figure 47: Normalized difference (max) in potential when compared to SH

Figures 46 to 49 show the RMS and max of the differences in potential and acceleration corresponding to each of the bands and for the various fidelity models. As
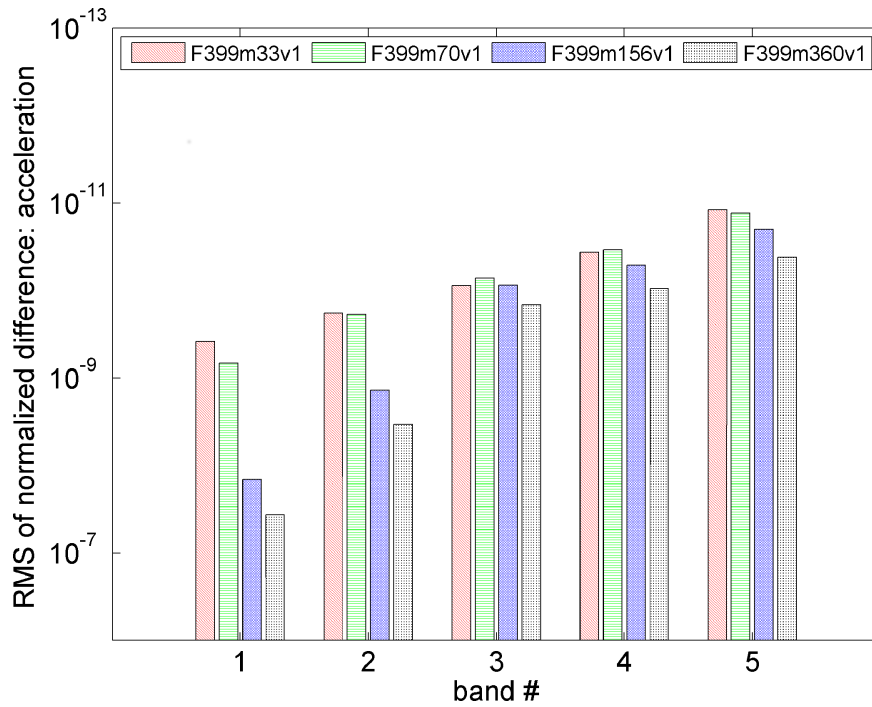
Figure 48: Normalized difference (RMS) in acceleration when compared to SH
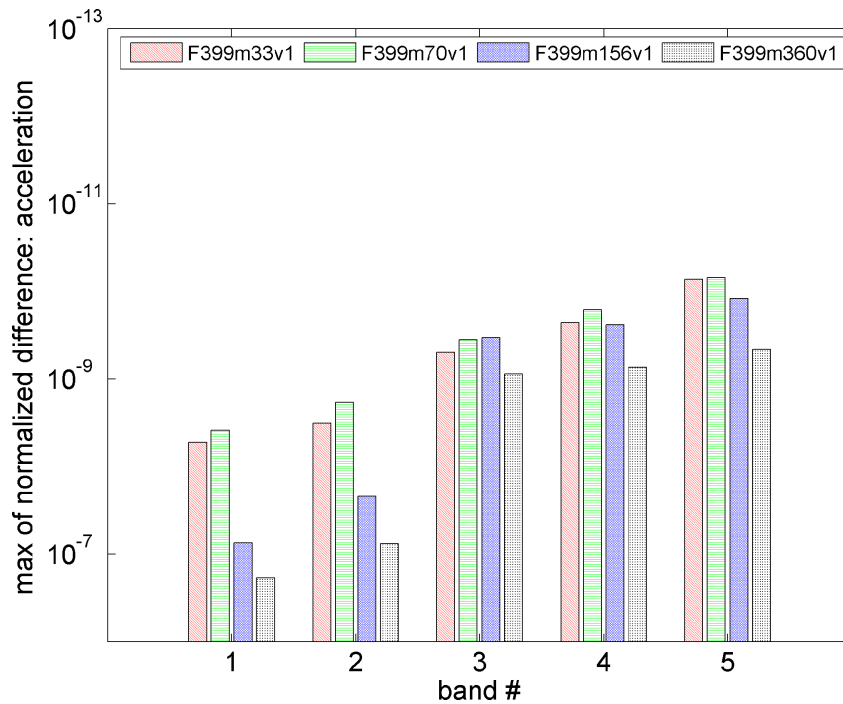
Figure 49: Normalized difference (max) in acceleration when compared to SH

expected, the RMS of the differences in potential and acceleration is always found to be less than the target residual from the fitting process and the max of the differences

is generally one order of magnitude greater. The observed differences in lower fidelity fields are much smaller due to the lower target residuals associated with the lower field fits.
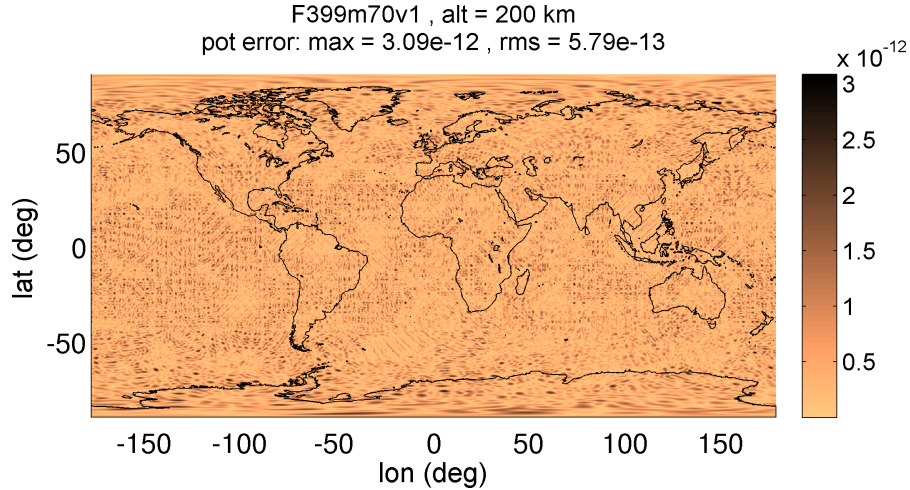
Figure 50: Potential difference profile, 200 km altitude:70×70 field
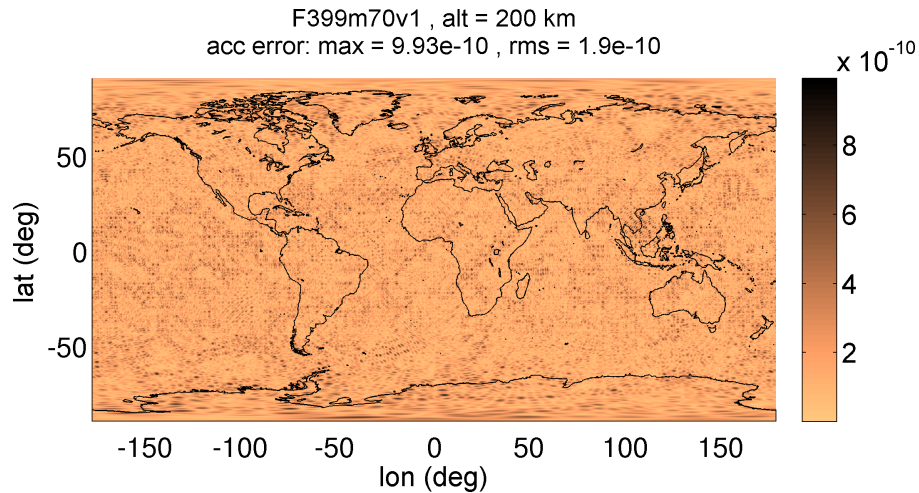
Figure 51: Acceleration difference profile, 200 km altitude:70×70 field

Figure 50 to 53 show the differences (in normalized units) in potential and acceleration at 200 km altitude for a Fetch model interpolating a 70×70 SH model and a 360×360, respectively. As expected, the maximum differences in the potential and acceleration are found to be less than their computed runtime residual tolerances. Looking at Fig. 37, the estimated RMS accuracies of the GGM03C 360×360 model
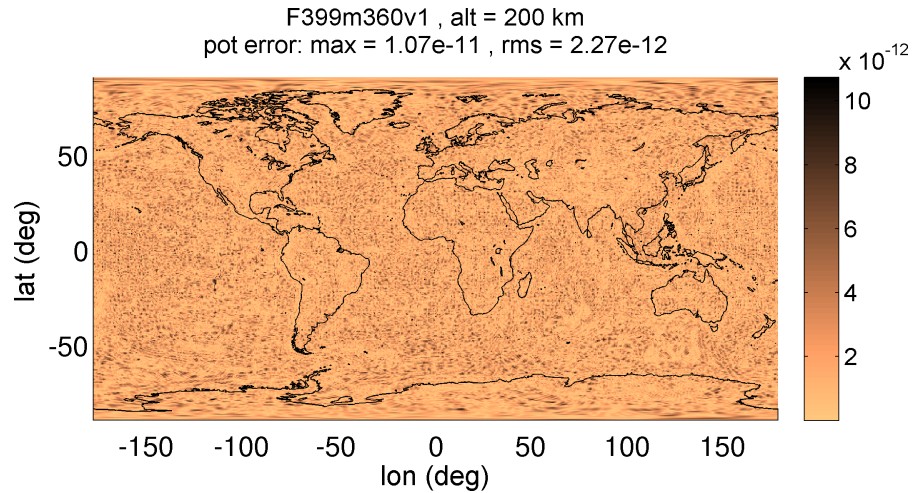
Figure 52: Potential difference profile, 200 km altitude:360×360 field
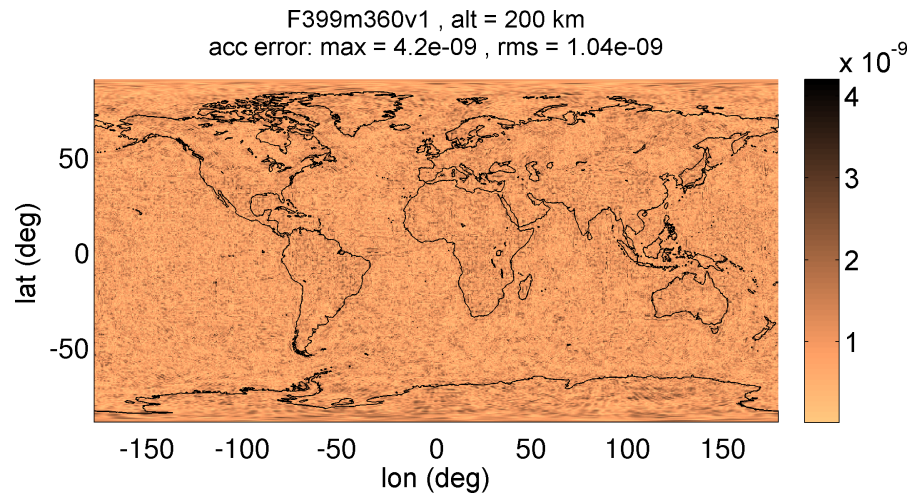
Figure 53: Acceleration difference profile, 200 km altitude:360×360 field

are approximately 3E-8 at the surface (and get mapped to a slightly lower value when evaluated at 200 km), therefore the Fetch model, with a max geopotential difference of 1E-11, is conservatively (noting the RMS of the difference is even lower) three orders of magnitude below the noise level of the SH function being fit. The Cubed-Sphere model is also three orders of magnitude below the noise of the SH model.[66] Therefore, both the Cubed-Sphere and Fetch models can be considered sufficiently accurate (if not overly accurate) with respect to the true geopotential.

Figures 54 and 55 shows the comparison to SH statistics for a complete sweep of
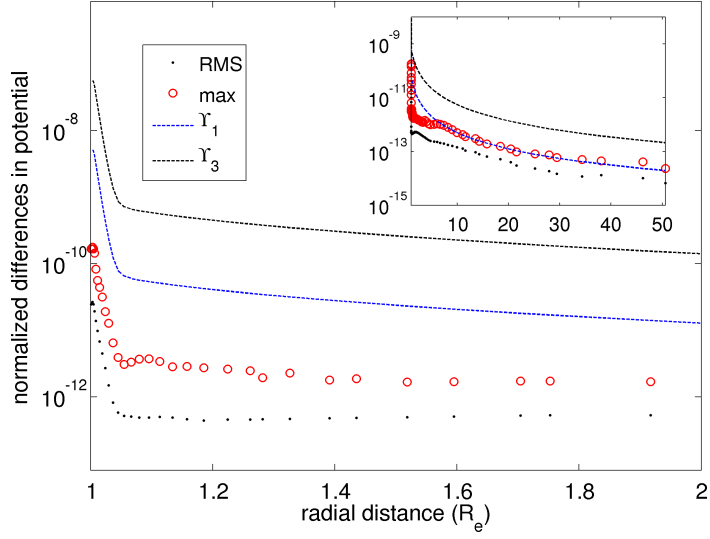
Figure 54: Potential normalized differences

the global domain for the F399m156v1 model. Each point corresponds to a shell and contains the max or RMS of the differences in potential and acceleration over all the cells, with 27 random evaluations within each cell. Near the surface, Υ_2 generally is the limiting one of the four residual criteria $(\vec{\Upsilon})$ and therefore the RMS of the differences in potential are significantly smaller in these regions.
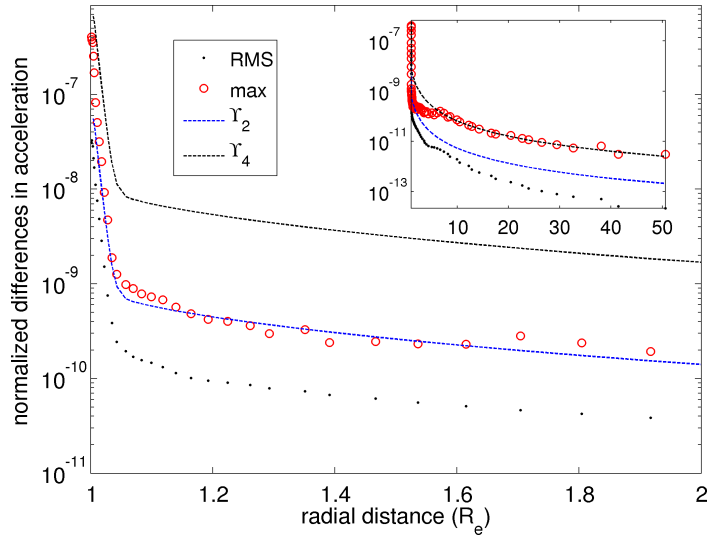
Figure 55: Acceleration normalized differences

3.7.2 Speed comparison

As Fetch trades memory for runtime performance, the speed at which memory (RAM) can be accessed has a major impact on the model's runtime performance. Randomized reads from memory are slower than sequential reads or reads from adjacent memory banks. Hence, the runtime performance of Fetch varies significantly, depending on how the coefficients are being accessed.

Fetch's runtime performance is independent of the SH degree and order. Figure 56 shows minimum and maximum absolute evaluation time vs number of coefficients. To generate the minimum evaluation time data, multiple points per cell are evaluated to mimic the near sequential access behavior. The maximum evaluation time data are generated by evaluating, in sequence, non-adjacent cells with the same number of coefficients. Given the computational nature of SH, its evaluation time increases quadratically as the degree and order of the SH field increases (see Fig. 57). From the timing data, we can compute the expected runtime speedup corresponding for a given number of coefficients. Higher order Fetch models demonstrate up to three to four orders of magnitude in speedup. It is noted that the Fetch runtime performance is cut in half in the regions where the primary and rotated grids overlap.

Fortunately, in most foreseen applications, the gravity field will be queried in a near sequential manner, such as a trajectory following a 1D path in the 3D domain. Hence we can expect absolute evaluation times close to the minimum evaluation time values in Fig. 56. Typical low altitude applications (those more likely to require high-fidelity gravity) will encounter cells with the greatest number of polynomial coefficients. The higher order derivative routines are also tested and it is found that including the Jacobian or the Jacobian and the Hessian of the acceleration cost an additional ~15% and ~25% compute time, respectively, when compared to the potential and acceleration only case. On the contrary, in a simple SH experiment, it is noted the additional Jacobian or Jacobian and Hessian calculations lead to 50% and
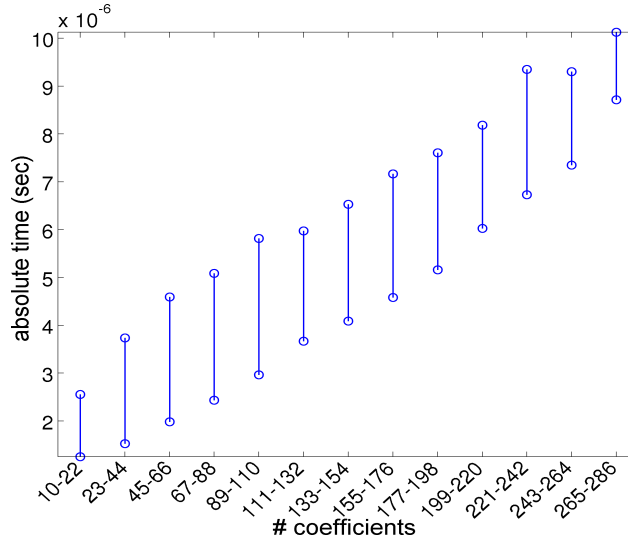
Figure 56: Fetch: absolute compute time, single sub-grid evaluation of potential and acceleration

200% compute time increases, respectively. Therefore, the speedups of Fetch model are further amplified in the case of higher order derivatives.

3.7.3   1D Path comparison

Table 15: Path classification

Path #	Type	Perigee alt. (km)	Apogee alt. (km)	Inclination (deg)	# Revs	Func. calls
1	Low altitude, circular, inclined	200	200	65	32.5	65,000
2	Low altitude, circular, near polar	500	500	85	30.5	46,000
3	Medium altitude, circular, near polar	1,350	1,350	85	25.6	39,500
4	High altitude, circular, near polar	4,050	4,050	85	16.4	24,700
5	Low perigee, highly eccentric, inclined	150	$5R_e$	65	6.5	12,600

To gauge the performance of Fetch in a realistic application, we query successive calls according to a 1D trajectory or path in the global domain. Because different paths access different cells with a different number of coefficients, five representative spacecraft paths (see Table 15 and Fig. 58) are considered. The timings of the Fetch and SH calls are evaluated for each of the four Fetch models and each of the five spacecraft trajectories. It is assumed that the spacecraft trajectories are precomputed and the timing results only include the Fetch and SH function calls.

Figure 59 summarizes the speedup results. For medium to high order Fetch models
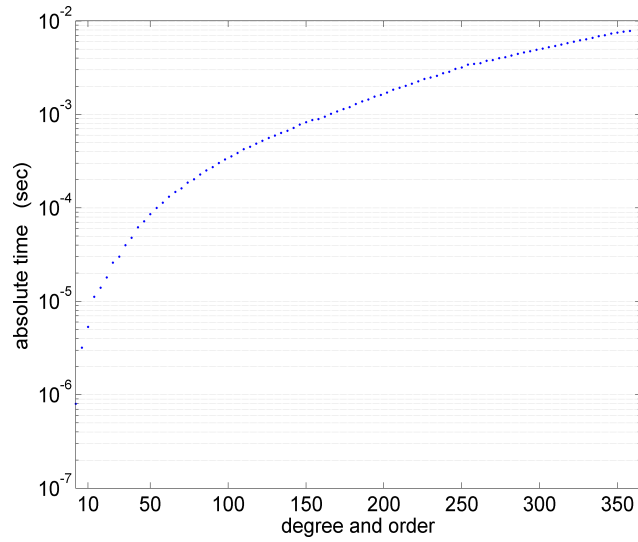
Figure 57: Absolute compute time for SH
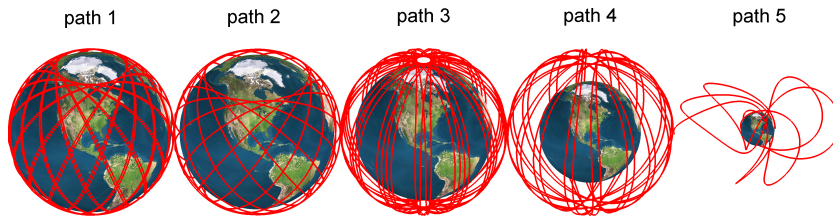
Figure 58: Various paths
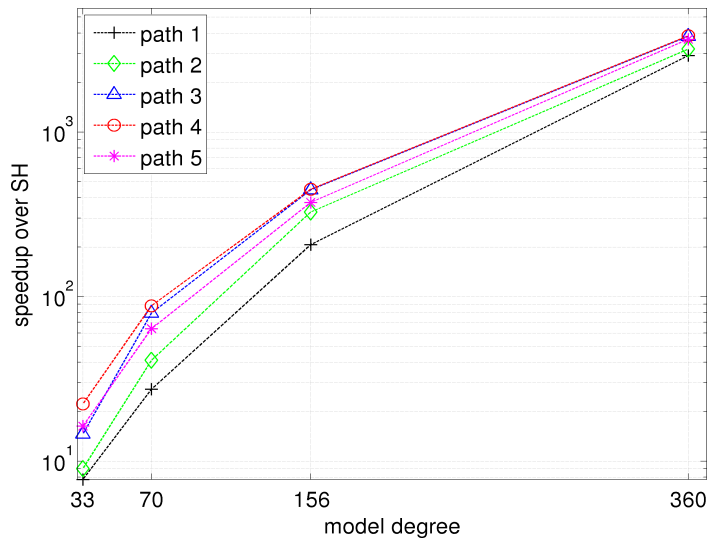
Figure 59: Fetch model speedups over SH: typical case of path evaluation such that the sequential calls to Fetch are from the same or neighboring cells.

we achieve multiple orders of magnitude in speedup. Specifically for the complete 360×360 SH field the final speedup varies from 2,900 to 3,900 times.

3.8 Chapter Conclusion

The main objectives of the Fetch model is to provide a fast and accurate way for calculating high order gravity fields while preserving the mathematical attractiveness associated with the SH formulation. The Fetch method trades an affordable memory investment for unprecedented speed gains. It uses a modified Junkins weighting function technique to achieve continuity over the whole solution domain and allows for localized resolution via adaptive polynomial fitting. The geopotential is the only interpolated function with the acceleration and any higher order derivatives calculated by explicitly differentiating the interpolant. The singularity present at the poles when conventionally dealing in spherical coordinates is tackled by implementing a two level grid structure with an overlapping latitude band. Therefore, the four priorities (continuous/smooth, adaptive, non-singular, accurate) laid out in the introduction are achieved. The memory footprint is fixed at the cost of storing the potential only, and scales almost linearly with SH degree and order. In contrast, the speedup scales in a near cubic fashion, making a compelling case for applications requiring high order SH.

Adaptivity for the Fetch model is achieved both through the radial direction and through the optimization of choosing the best out of 228 candidate interpolants for each node. Precomputed analytic solutions to the least squares normal equations afford the extreme flexibility of evaluating all candidate interpolants for all nodes (up to 8 million in this study). Furthermore, target residual tolerances for each node are calibrated based on the published accuracy of the GMM03C gravity model. These residual levels are used to guide the fitting process, so as to minimize the number of coefficients needed for each node while maintaining a uniform global residual profile

that is consistent with the physics of the problem and the accuracy of the SH function being fit. A master-worker based parallel version of the algorithm is implemented (in MPI) to efficiently generate the coefficients for the high order SH fields. The parallel coefficient generation algorithm enables the fitting of high degree SH fields.The highest resolution currently investigated (and native size of the GGM03C model) is 360×360 where the Fetch model achieves up to 3,900x speedups. The residual profile for the potential and acceleration are uniform according to the order of the tolerance specified. In its current form, the Fetch model does not satisfy the Laplacian equation. Future works could consider either constraining the least squares problem or choosing basis functions that do naturally satisfy the Laplacian equation.

The ease of implementation combined with speed and favorable continuity properties make the Fetch model attractive for high-fidelity orbit determination or trajectory optimization tasks. Chapter 6 showcases the Fetch model applied to the problem of simulating multiple spacecraft trajectories on the GPU. The Fetch runtime code is embedded as part of this thesis in appendix E.

CHAPTER IV

EPHEMERIS COMPUTATION

4.1   Chapter Summary

Precise trajectory simulations typically require an ephemeris retrieval system, i.e. some mechanism to identify planetary body states and orientations at given times. However, the ephemeris systems most commonly used throughout industry and academia are, by design, general in their capabilities and application. In this chapter a new system called FIRE (Fast Interpolated Runtime Ephemeris) is introduced. FIRE is designed for custom trajectory applications that favor speed and smooth derivatives. The new system minimizes the overhead associated with ephemeris calls through the use of archived splines, a runtime ephemeris (stored in random access memory of the computer), and batch processing routines. Further, our approach naturally provides first and second time derivatives for a small additional computational cost. The derivative capability is particularly attractive for optimization and targeting where smooth and accurate derivatives are important. Relative performance comparisons with the Jet Propulsion Laboratory's Spacecraft Planet Instrument C-matrix Events (SPICE) ephemeris system show typical speed improvements of up to two orders of magnitude for various state and orientation calls. Performance comparisons for high-fidelity trajectory propagations are also considered and a factor of 60 in performance increase is achieved for typical cases. The proposed tool has potential value to any high precision application or software requiring fast, accurate, and smooth ephemeris data.

4.2   Chapter Nomenclature

$a, e, i, \omega, \Omega, \nu$ Classical orbital Elements

N	Number of knot pairs
ζ	State vector
ns	Size of state vector
V	Interpolated variable at each knot
ξ	Knot to current time distance
h	Knot to knot distance
t	Current time
$t0$	Epoch time
M	Scaled second derivatives at knots
i	Current knot index
x, y, z	Components of position vector
u, v, w	Components of velocity vector
G	Standard gravitational parameter
m	Mass of the body
R	Rotation matrix (3×3)
α, β, γ	1st, 2nd and 3rd Euler angles
JD	Julian Date
Nr	Normalizing Factor
wrt	With respect to
NPR	Number of points per revolution
EMS	Earth-Moon-Sun
FFT	Fast Fourier Transformation
ACE	Archived Cubic spline Ephemeris
RACE	Runtime Adaptive Custom Ephemeris
JMSS	Jupiter-Moons-Saturn-Sun
FIRE	Fast interpolated runtime ephemeris
SPICE	Spacecraft Planet Instrument C-matrix Events

4.3   Introduction and Background

In the last chapter a new high performance, higher order gravity model was proposed. The next bottleneck problem commonly encountered when computing high-fidelity perturbations is that of computing solar system body ephemeris. The solar system body state and orientation data (commonly computed via JPL's SPICE model[2]) has applications in a variety aerospace applications.[63,46,113,114,3,83]   The SPICE model represents the current state-of-the art and provides a broad level of capability apart from computing body state and orientations.It is well known that SPICE ephemeris

calls suffer from large overheads and are one the major speed bottleneck for precise applications. A large amount of computational resources are wasted in trajectory simulations when applied to problems like optimization, differential correction, orbit determination, and Monte-Carlo analysis.

With this fact being the motivation, a new custom ephemeris system is proposed in this chapter. The new system, called FIRE (Fast Interpolated Runtime Ephemeris), maintains the heavily relied upon accuracy, yet eliminates or substantially reduces the typical computational burdens associated with ephemeris calls. FIRE is particularly suited for problems that favor higher speeds and smooth derivatives, such as trajectory optimization,[110,109,60,118] orbit determination[25,114,72,83] and Monte Carlo sensitivity analysis.[35,3] The new system generally favors speed in sacrifice of relatively more memory. The basic structure consists of an internally formatted, archived ephemeris called ACE (created from an already established ephemeris such as SPICE) and a problem dependent, dynamically created runtime ephemeris called the RACE. Implementing a runtime ephemeris stored directly in RAM significantly reduces the overhead associated with the multiple runtime ephemeris calls.

Various numerical techniques such as a fast and efficient cubic spline interpolation for ACE generation, FFT for identifying base frequencies, batched runtime calls (batch calls) to reduce overheads, adaptive tree structure evaluation to eliminate redundancy, and numerically stable floating point algorithms have been implemented. FIRE also provides the user with continuous and analytic first and second time derivatives for all the ephemeris states (position, velocity) and orientation matrices. This valuable derivative feature is absent from SPICE routines (like "SPKEZ","SPKEZP" and "PXFORM"), where derivatives are available only through the expensive and less accurate numerical differencing method.

For this thesis, we use SPICE as the benchmark for our performance comparison as it is arguably the most widely accepted ephemeris retrieval architecture publicly

available. Extensive performance comparisons for direct position and velocity calls show that FIRE is ∼70 to ∼250 times faster (depending upon the number of bodies and type of call). If only the positions of bodies are evaluated, then FIRE performs ∼44 to ∼197 times faster. Further, a performance increase of ∼150 to ∼250 times is observed if we also compute the orientation matrix along with the position and velocity vector. When implemented in trajectory integration problems, the FIRE integration is demonstrated to be ∼50 to ∼70 times faster than the same integration using SPICE. For similar applications using SPICE, there is little motivation for improving general algorithm efficiency because ephemeris calls dominate such a large fraction of the total computational burden. In the case of FIRE, users are free to experience the full benefit of improving the efficiency of their custom applications.

It should be emphasized that the FIRE system is not intended to be a replacement for the full functionality of SPICE or any other established ephemeris system. It acts as a wrapper over an already established ephemeris like SPICE and is intended to replace only the state and orientation ephemeris calls for custom applications that may benefit from FIRE's speed and smooth derivatives.

4.4   FIRE Architecture and Core Numerical Computation

This section gives an overview of FIRE's main architecture and its core routines. The FIRE architecture is based upon the premise of increasing speed and maintaining accuracy while sacrificing memory as efficiently as possible. It is written in Fortran using a modular approach to ease its implementation and expandability.The main architecture can be broadly divided into three sub-systems:

1. Archived Cubic spline Ephemeris (ACE)

2. Runtime Adaptive Custom Ephemeris (RACE)

3. RACE loading and Runtime batch processing routines

The above mentioned sub-systems are computationally separate from each other, each containing its own set of local core routines. A standard input method from a text file corresponding to a "namelist" local to that system has been followed, thereby imparting robustness and flexibility to the code. Even though the SPICE naming convention has been adopted for consistency, we can create our own naming convention and our own custom bodies or spacecraft by changing the relevant inputs. Note, however, for the tree navigation algorithm, we do require that the "tree trunk' is the main solar system barycenter defined by a body number of zero (see Fig. 60). Details on the tree structure are discussed later.
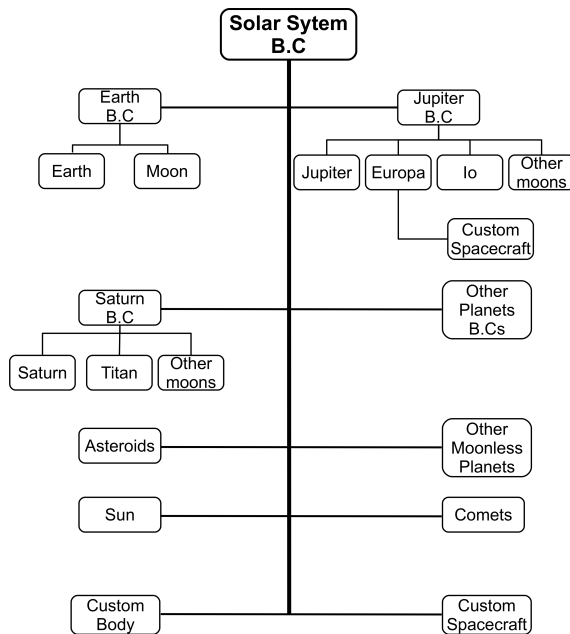
Figure 60: Sample Tree Diagram

4.4.1 Archived Cubic Spline Ephemeris (ACE)

4.4.1.1 ACE computation

Cubic spline interpolation is actively being used in various fields like Robotics,[62,58] Numerical integration,[126] Image Interpolation[137] and Animation.[124] The ephemeris states of spacecraft and planetary bodies are well behaved functions. Hence, cubic

splines can easily achieve the desired accuracy and efficiency required for interpolation. Further, cubic splines naturally provide continuous derivatives up to second order. The second derivative will have kinks at the knots but remain continuous while the first derivatives are smooth and continuous across the full domain. Note that we interpolate on velocities separately instead of using the first derivative of the position in order to maintain accuracy and continuous second derivatives. We choose the cubic splines primarily because they are remarkably fast to compute in comparison to other popular, higher order splines such as Chebyshev polynomials and B-splines.

Even though SPICE is already based on a Chebyshev polynomial, we interpolate on the interpolated ephemeris for ease of implementation. We note that errors in our interpolation of SPICE are on the same order or less than those interpolation errors published by SPICE. Nonetheless, a possible future works could include interpolating the raw currently unpublished data that SPICE uses for its interpolation.

ACE is generated by implementing an efficient cubic spline algorithm to interpolate the data generated by an established ephemeris. We adopt a clamped cubic spline method well suited for accuracy and smoothness.[34] The end conditions require the first derivative of the interpolation variable at the first and the last knot. These derivatives, for each state and orientation angle being interpolated, are calculated using a fourth order numerical difference scheme. For spline computation, the linear, tridiagonal system is given by Eq. 78.

$$\mathbf{A}\vec{M} = \vec{V} \tag{78}$$

where A is a tridiagonal matrix given by Eq. 79:

$$\mathbf{A} = \begin{bmatrix} 3.5 & 1 & & & & & 0 \\ & 1 & 4 & 1 & & & \\ & & 1 & 4 & 1 & & \\ & & & . & . & . & \\ & & & & 1 & 4 & 1 \\ 0 & & & & & 1 & 3.5 \end{bmatrix} \tag{79}$$

The linear system given by Eq. 78 is solved for $M(1)$ to $M(N-1)$ by using a numerically stable tridiagonal matrix algorithm.[30] Further, $M(0)$ and $M(N)$ are evaluated using the end conditions from Eqs. 80 and 81. Note: $M(1)$ to $M(N-1)$ values are solved first and then $M(0)$ and $M(N)$ are evaluated, subsequently.

$$M(0) = \frac{3}{h}[\frac{(V(1) - V(0))}{h} - \dot{V}(0)] - \frac{M(1)}{2} \tag{80}$$

$$M(N) = \frac{3}{h}[\frac{(V(N) - V(N-1))}{h} - \dot{V}(N)] - \frac{M(N-1)}{2} \tag{81}$$

After computing $M(0)$ to $M(N)$, the spline coefficients (a, b, c, d) are evaluated using the Eqs. 82- 85

$$a = V(i) \tag{82}$$

$$b = \frac{(V(i+1) - V(i))}{h} - \frac{h}{6}(2M(i) + M(i+1)) \tag{83}$$

$$c = \frac{M(i)}{2} \tag{84}$$

$$d = \frac{(M(i+1) - M(i))}{(6h)} \tag{85}$$

Finally the spline is evaluated using a numerically efficient form given by Eq. 86 (where ζ is a dummy interpolation variable or state vector) and 1st and 2nd derivatives are calculated by Eq. 87 and Eq. 88, respectively.

$$\zeta = a + \xi(b + \xi(c + \xi d)) \tag{86}$$

$$\dot{\zeta} = b + \xi(2c + 3\xi d) \tag{87}$$

$$\ddot{\zeta} = 2(c + 3\xi d) \tag{88}$$

The distance between knots, while performing interpolation, directly affects the accuracy and smoothness of the interpolation.[34] In our case, the distance between knots is the minimum time step required for successful interpolation which is driven by the frequency of the body having the smallest orbital period for a particular system of related bodies. Take the Jupiter system as an example containing the Jupiter barycenter, Jupiter itself, and its major moons. Io has the smallest period of all the major satellites. In this case, the frequency associated with Io will of course be observed in the ephemerides of all bodies in the Jupiter system. Therefore, to capture the high resolution motion, we assign all bodies a time step commensurate with Io. In fact, to obtain the final time step for the whole Jupiter system, we divide Io's period by NPR (number of points per rev), a user defined accuracy parameter. In our experiments, we find that NPR values in the range of 30-60 gives good results.

This time step corresponding to the knot separation is given to all the moons of Jupiter according to the "$MINGM$" criteria. The "$MINGM$" criteria allows a body to be considered if its GM value is above a predefined value. Decreasing "$MINGM$" incorporates smaller bodies in the time step computation hence increasing the accuracy of interpolation. A body with GM less than the "$MINGM$" is assumed to have no effect on the states of the remaining bodies in the system. Therefore its frequency signature is deemed unimportant to the dynamics of the problem. As we will discuss later regarding the rotation states, the FFT method is an alternative approach to choosing the knot separation.

The ability to attain greater accuracy by either increasing the number of points or by decreasing the $MINGM$ defined value can be misleading. For smooth data, if the knots are too densely populated, then the interpolating polynomial will use
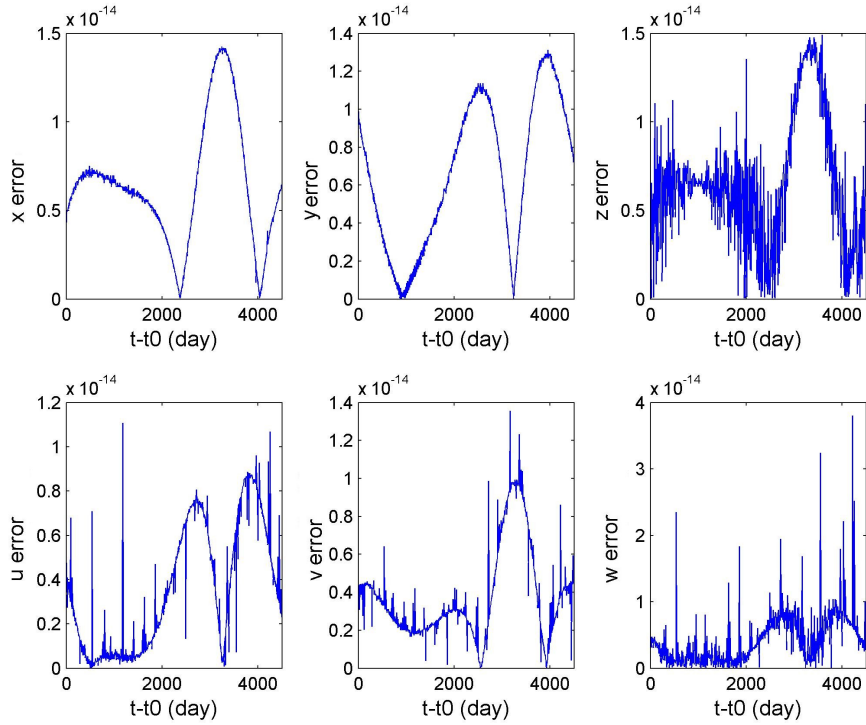
Figure 61: Normalized error in Jupiter's Barycenter states

its high-degree coefficients, in combination with large and almost precisely cancel-ing combinations. This may cause the interpolating polynomial values to oscillate between its constrained points, and hence may produce spikes which affect the well boundedness of errors.

The accuracy of the spline derivatives is typically less than the accuracy of their immediate integrals. Hence, the normalized accuracy for the zeroth, first and second derivative is generally on the order of 1E-8(normalized dimensionless value), 1E-6 and 1E-4 respectively for bodies and is of the order of 1E-14 (normalized dimensionless value), 1E-11 and 1E-7 for barycenters. The normalizing factor (Nr) is calculated as per Eq. 89. For our purposes, it is very important that the zeroth derivative be accurate. Figure 61 shows a representative error plot for position and velocity components along with their first and second derivatives for Jupiter's barycenter.

$$Nr = max(\zeta_i) \quad \forall\, i = 1 \ldots ns \tag{89}$$

Note that the SPICE data, is not necessarily smooth to second order across the full time domain. Discontinuities in derivatives may exist at junctions in the SPICE interpolating functions. While the "error" plots presume that the SPICE numerical derivatives are truth, one could argue that the FIRE derivatives are closer to the truth in the absence of numerical differencing. In fact, for numerical purposes regarding targeting and optimization, it is more important for the derivatives to be self consistent rather than absolutely correct.

We choose Euler angles for orientation interpolation because they perfectly preserve orthogonality in the resulting rotation matrix. Other methods such as Quaternions,[125] and Rodrigues Parameters[134] were investigated extensively in order to avoid the computationally expensive trigonometric calls. However, it is well known that interpolating quaternions introduces complications including sign ambiguities and or loss of orthogonality.[124]

Note that the SPICE interface with orientation data is a common rotation matrix between user defined frames. Fire is currently limited to obtaining rotation matrices between the IAU defined body fixed coordinate systems and the base ecliptic J2000 inertial frame.[121] A rotation matrix between two general frames is achievable with FIRE using two separate rotation calls and a matrix transpose and multiplication.

FFT's can be easily used to catch frequencies of a periodic system.[31] In this study, an efficient and numerically stable FFT algorithm[57,23] is used to calculate the minimum step size required for interpolation of Euler angles. For a given angle, the frequency corresponding to the largest spike in the Fourier transformed space is recorded and the corresponding time step is calculated. This provides us with a reliable, robust, and efficient interpolation method, even for complex and subtle body dynamics. A representative FFT plot for the three Euler angles of the Moon are shown in Fig. 62.

Here, α and β represent the pointing direction of the north pole of the body in
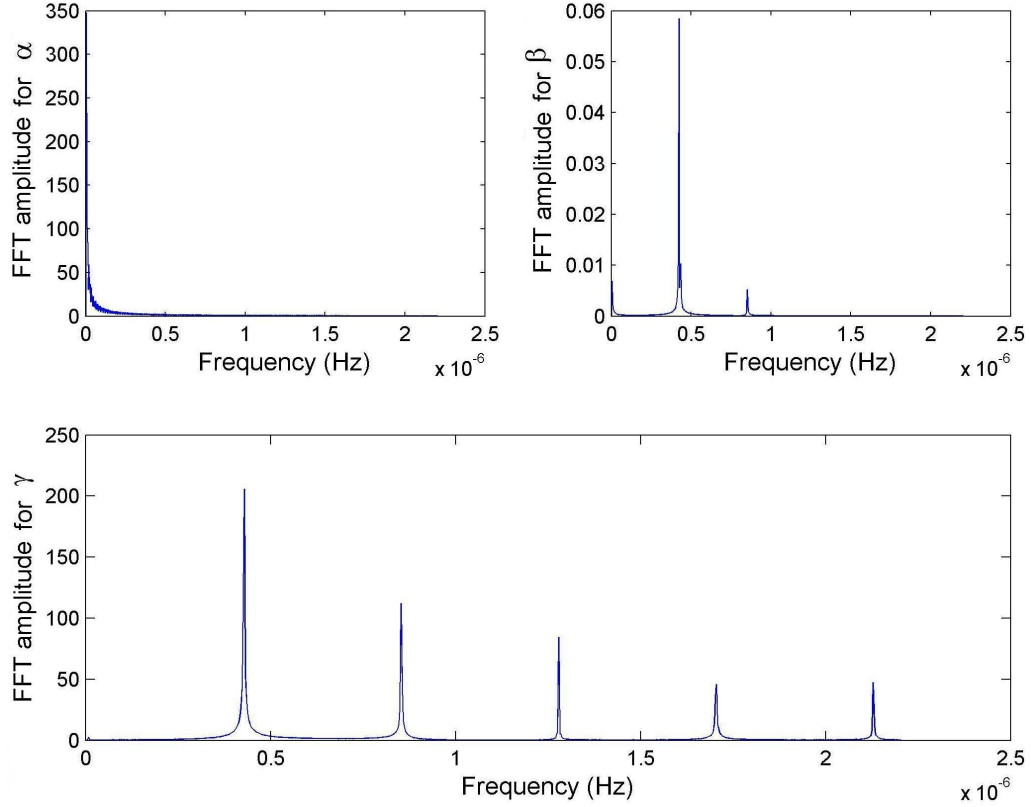
Figure 62: Fast Fourier Transformation for Moon's three Euler angles

question, while γ represents the angular location of the prime meridian. The accuracy of the interpolation is of the order of 1E-15 (non-dimensional error) for both the slow period angles and 1E-13 (nondimensional error) for the fast moving angle. Relative normalized error plots for Euler angles of Titan (Neptune's largest moon) for 50 days are shown in Fig. 63.

A $3 - 1 - 3$ rotation matrix (R) has been adopted via IAU Standards.[121] The expression for the time derivatives for R in terms of the time derivatives of α, β, and γ are straight-forward. We obtain the time derivatives of the states directly from the interpolation. The actual implementation for the time derivative of R is optimized for serial access, favoring speed and re-use of data. Note that currently the FIRE system is limited only to natural body orientation data retrieval.
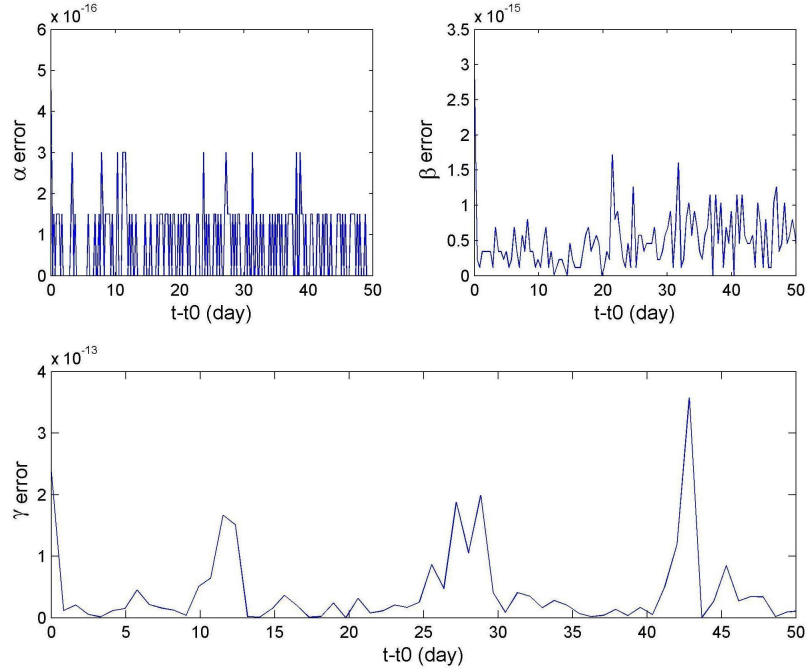
Figure 63: Relative normalized error in Titan's Euler angles

4.4.1.2   ACE storage

ACE is stored in a structured format, consisting of numerous gravitational subsystems. Each body of a subsystem, has up to 3 associated binary files: one for position, one for velocity, and one for rotation. The binary files contain the dependent variables and their second derivatives at all the knot points for each interpolation variable. The spline coefficients are calculated at runtime using optimized implementations of equations 5-8. The independent knot (time) values are stored in an efficient manner to avoid redundancy while preserving speed. An archived ephemeris (ACE) is typically generated when using FIRE for the first time or when there is major update in the underlying ephemeris system used to generate ACE.

4.4.2   Runtime Adaptive Custom Ephemeris (RACE)

RACE is a binary, dense array, whose main purpose is to significantly increase the speed of run-time ephemeris state and orientation calls by eliminating the overhead

arising from invoking the archived spline. RACE also provides the user with a portability option to store as a binary file the problem specific RACE for reuse later. Hence, multiple RACE's can be generated and stored for different classes of problems. This strategy leads to the idea of parent (ACE) and child (RACE) ephemerides.

By default, each body within the RACE has its state relative to its own native barycenter. A user defined input can modify the RACE to change the interpolation center for the bodies present. Typically we choose the new interpolation center as the center of integration for a trajectory propagation and may lead to 50% to 150% performance improvement at runtime. The memory requirement of the runtime ephemeris typically varies from 1 to 100 megabytes (approximately) depending upon the accuracy of ACE and the time span and number of body states required for the problem.

The number of knots for all the bodies is equal to some 2^k value (where k is an integer). Hence, knowing the body and the corresponding particular interpolation variable having the largest value of k provides a mechanism to uniquely identify the appropriate knots for each state and orientation in a batch call without performing expensive searches in the RACE at runtime. This algorithm requires one "double" to "integer" conversion and just a few elementary floating point operations, instead of a linear or a binary search algorithm.

4.4.3 RACE loading and Runtime batch processing routines

The third sub-system is invoked at runtime and performs the following two operations:

1. Dynamic allocation of the RACE in memory

2. Computing the state vectors, the rotation matrices and any derivatives by evaluating the tree navigation algorithm (see Fig. 60)

In contrast to the capabilities of SPICE, the new system employs a single batch call for interpolation of all of the necessary states and orientations at a particular

time[1]. The term "batch call" refers to calling all the states of multiple bodies in one call rather than calling the routines over and over for each body state and orientation. These batch calls significantly reduce the overhead associated with the common initialization for each body.

In cases when an ephemeris is used in the force models for optimization and or targeting routines, continuous first and second time derivatives may be necessary for positions, velocities, and rotation matrices. Keeping this in mind, FIRE also provides the user with set of flagged routines which can be invoked to get the first or both the first and the second derivatives of position, velocity, rotation, or any combination of these via only one batch call. This derivative data can be extremely useful for problems which are sensitive, like trajectories with multiple close flybys or multiple gravitating bodies.[76, 141] The RACE is general and users are free to call for states and derivatives relative to any center.

Navigation of the tree structure (see Fig. 60) is required at runtime if the user requested reference center is different from the RACE interpolation center. In the case of trajectory integration applications, this reference center is typically the problem specific center of integration. Again for a general system like SPICE, there is overhead associated with evaluating the tree at runtime for every call and every body. In the FIRE architecture, the batch calls provide all the necessary states with respect to one common center. A efficient tree navigation algorithm is implemented, which removes any redundant calls without sacrificing the numerical stability and accuracy of the state being calculated.

The overview of the complete FIRE system is given in Fig. 64. Details on FIRE implementation and usage are given in appendix D.

---

[1]We note that the Matlab version of SPICE, called MICE, includes batch calls although we suspect it is simply a wrapper around the un-batched SPICE.
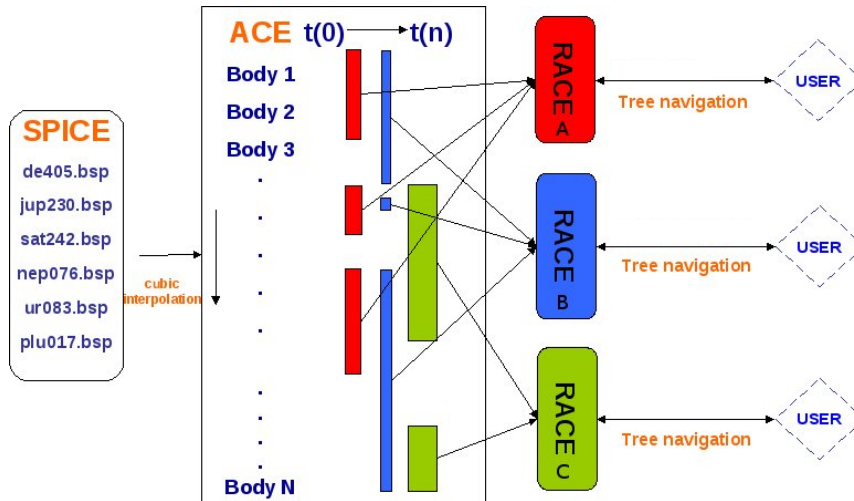
Figure 64: Overview of the FIRE system

4.5   Performance

JPL's ephemeris generation system (SPICE), is used for benchmarking the performance of the FIRE system. The system configuration used for benchmarking is an Intel dual core processor (2.6Ghz) with 4 GB of RAM. The benchmarking is done in two ways. First, we directly call the routine for a certain fixed number of calls and record the relative performance gain. Second, we perform two typical high-fidelity trajectory simulations for the Earth-Moon-Sun (EMS) system and the Saturn system with Saturn and its nine moons along with Jupiter and the Sun. We take into account accelerations due to third body perturbation and gravity field of the central body (via spherical harmonics).

The code is compiled and linked on the Intel Fortran Compiler for Windows version 11.0 and is subjected to default "-fast" compiler optimizations.

4.5.1   Performance with direct routine calls

In this comparison $3*2^{24}$ (approximately 50 million) calls with random time inputs are used to evaluate the performance of SPICE and FIRE. We use the SPICE routines "SPKEZP, SPKEZ" for computing position and velocity and "PXFORM" with

Table 16: Various cases for Earth-Moon-Sun performance comparison

System	Earth-Moon-Sun:
Type1	Moon wrt Earth-Moon barycenter(native barycenter call)
Type2	Earth and Moon wrt Earth-Moon barycenter(native barycenter call)
Type3	Earth wrt Moon(native system call)
Type4	Earth and Moon wrt Solar-System barycenter(half tree evaluation)
Type5	Earth, Earth-Moon barycenter and Sun wrt Moon(typical call)
Type6	Earth wrt Sun(full tree evaluation)

Table 17: Various cases for Jupiter-Moons-Saturn-Sun performance comparison

System	Jupiter-Moons-Saturn-Sun:
Type1	Jupiter wrt Jupiter barycenter
Type2	Io, Europa, Ganymede, Callisto wrt Jupiter barycenter
Type3	Io, Europa, Ganymede, Callisto wrt Io
Type4	Io, Europa, Ganymede, Callisto wrt Solar-System barycenter
Type5	Io, Europa, Ganymede, Callisto, Jupiter, Sun, Saturn barycenter wrt Europa
Type6	Jupiter wrt Saturn barycenter

integer inputs for generation of the rotation matrix.

Two gravitational systems are studied for this comparison: the Earth-Moon-Sun (EMS) system and the Jupiter-Moons-Saturn-Sun (JMSS) system. The EMS system has only three bodies passed at once, hence highlighting the performance gain due to the use of RACE, the runtime matrix. The JMSS system has eight bodies and thereby quantifies the performance of the full capability of the tree navigation algorithm, batch calls, and RACE. A type defined bar graph system is used to illustrate the speedup of FIRE over SPICE. As shown in Table 16 and 17 there are a total of six types of ephemeris calls, each with different performance characteristics due to different path lengths required in evaluation of the tree algorithm (Fig. 60).

From Figs. 65, 66, 67, 68, we demonstrate dramatic performance gains from FIRE in comparison to SPICE. Note that the most expensive call requires a full navigation of the tree algorithm. Even in this worst performing case, FIRE routines are $45 - 65$ times faster than SPICE. The typical FIRE call where the batch routines are fully exploited and most of the bodies share a common interpolation center is demonstrated to be as much as 250 times faster.

Often a typical routine call includes the necessary position and velocity data for
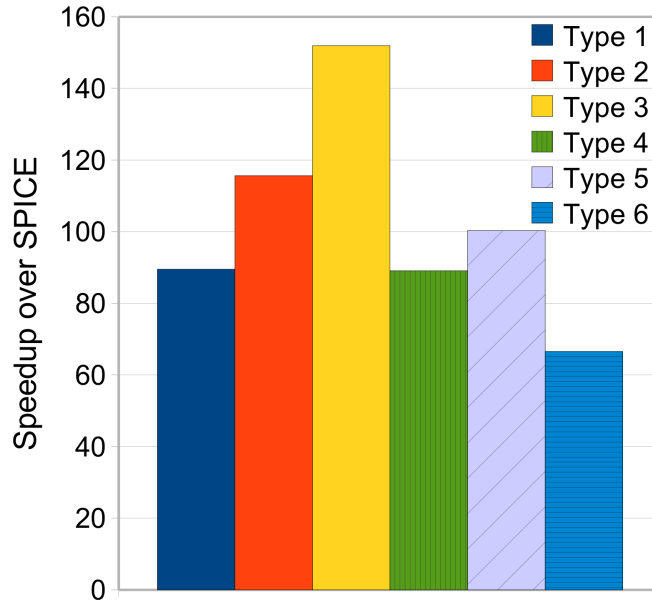
Figure 65: Speedup factor for EMS system (pos)

Table 18: Typical performance comparison ratio table for Earth-Moon-Sun system

Type	Frame/Center	SPICE time (s)	FIRE time (s)	**Speedup**
Typical call(pos and rot)	EclipJ2000/Earth	3066.63	20.17	**152**

all the bodies along with rotation data for some bodies. Hence, to demonstrate full performance capability of FIRE, these calls have been again evaluated along with rotation matrices for Earth in the first system and for Io in the second system. Also, we include a test with only rotation calls to a couple of bodies in the Jupiter system for completeness. Accordingly, Tables 18 and 19 show the performance for both the tests respectively. Hence for the FIRE performance the typical call is around $150-200$ times faster when rotation states are included. We note that the huge improvements when including rotation states may be over-shadowed by the heavy computational burdens of spherical harmonics or polyhedral potential calculations that typically accompany orientation state needs (see chapter 3 and 6).

A distinguishing feature of the FIRE system is its ability to provide the first and second time derivatives of all the ephemeris states. The states plus first derivative calls perform nearly equal to position-velocity calls only, and the combined states plus
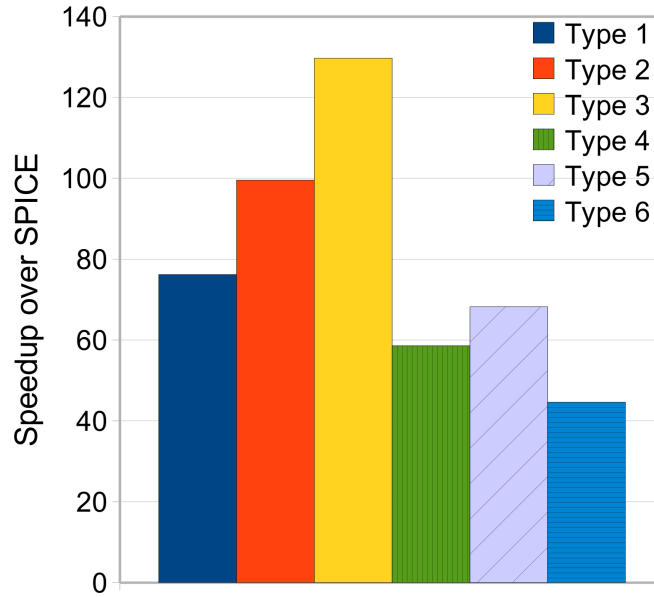
Figure 66: Speedup factor for EMS system (pos-vel)

Table 19: Typical performance comparison ratio table for Jupiter-Moons-Saturn-Sun system

Type	Frame/Center	SPICE time (s)	FIRE time (s)	**Speedup**
Typical call(pos and rot)	EclipJ2000/Io	5385.97	26.87	**200**
Rotation call only	EclipJ2000	3443.91	14.11	**244**

first and second derivative calls require approximately 1-5% more time than the first derivative only calls. We emphasize again that numerical derivative computation using SPICE requires additional state calls (typically 2 calls for central difference approximation), and the resulting derivatives are substantially less accurate due to round off errors.

SPICE is also compared with the DE405 customized routines by Miles Standish,[128] called PLEPH. It was found out that generally PLEPH (using typical maximum speed optimizations) is approximately 6 times faster than SPICE. It should be noted that PLEPH only works with DE405 ephemeris data, and thereby has a limited scope. Further, a recent study by a company called Astrodienst [1] also shows that the

---

[1] ftp://ftp.astro.com/pub/swisseph/doc/swisseph.pdf
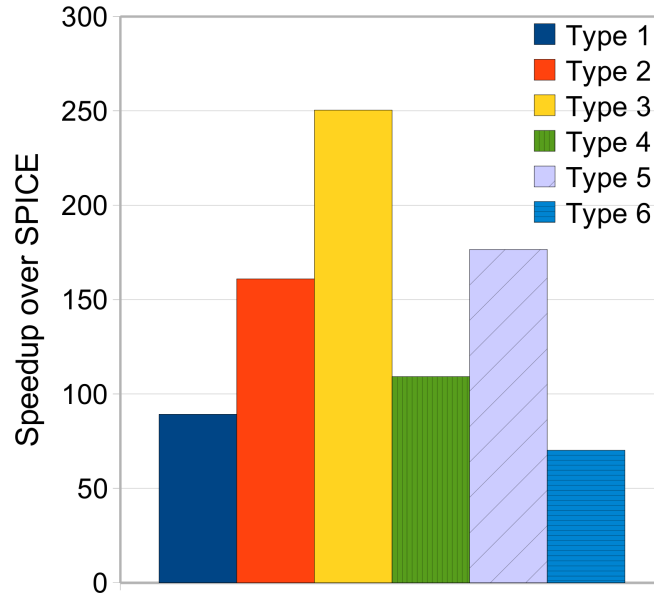
Figure 67: Speedup factors for JMSS system (pos)

semi-analytic ephemeris developed by Steve Moshier,[89] which enjoys no memory storage, is about 10 times slower than SPICE. The study also propose a new ephemeris called "The Swiss Ephemeris" [1], which significantly reduces the memory storage requirement of SPICE DE406 ephemeris without impacting its accuracy by more than 1 milli-arcsecond.

4.5.2 Performance during trajectory integration

Beyond the raw performance of using direct calls only, we now proceed to a more realistic comparison like that of spacecraft trajectory propagation. The relative performance gains will of course not be as impressive as the previous cases because of the integration and force calculation being done alongside ephemeris calls.

Two trajectory propagations are considered. One for the Earth-Moon-Sun system and the other for the Saturn-Moons-Jupiter-Sun system. Spherical harmonics gravity fields of various resolutions about the central body are also considered. For integration, a variable step 8th order with 7th order error control, Dormand and Prince

---

[1]http://www.astro.com/swisseph/
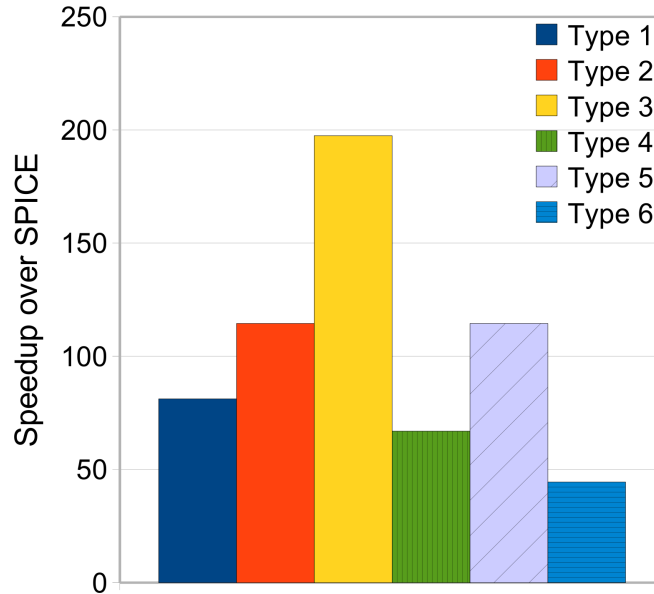
Figure 68: Speedup factors for JMSS system (pos-vel)

integrator[36] set to a normalized tolerance of 1E-14 is used. Ephemeris rotation calls are only included during the propagation when the non-spherical gravity field acceleration is included.

4.5.2.1 *Earth-Moon-Sun (EMS)*

For this comparison, a trajectory propagation of a mid altitude Earth orbiter has been performed taking into account the third-body perturbing effects of Moon and the Sun. This integration is performed for a period of 100 days. The initial conditions for this propagation are listed in Table 20. The size of RACE for this case was approximately 6 megabytes. Various Earth gravity fields evaluated in the propagation are given in Table 21.

Table 22 shows the comparison between FIRE and SPICE for different types of acceleration. Accuracy and performance of the numerical integrations are the two main criteria for comparison. Results of the trajectory integration show that FIRE performs 40 to 60 times faster than SPICE for typical cases (third body perturbations + J2-J10 terms). Also, as expected the speedup decreases as the resolution

Table 20: Initial condition (body-fixed frame at epoch) for EMS trajectory

Orbital Parameter	Value
Semi-major axis (a)	7500 (km)
Eccentricity (e)	0.074
Inclination (i)	35 (deg)
Argument of periapsis (ω)	9 (deg)
Longitude of ascending node (Ω)	20 (deg)
True anomaly at epoch (ν)	0 (deg)
Epoch (t0)	2454477.50 (JD)

Table 21: Gravity Field at Earth

Case	Acceleration combination
1	two body
2	2 by 0 (J2 only) + two body
3	10 by 0 (J2-J10) + two body
4	50 by 0 (J2-J50) + two body
5	10 by 10 (10 degree + order field) + two body
6	50 by 50 (50 degree + order field) + two body

Table 22: Performance comparison ratio table for EMS trajectory propogation

Cases →	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
SPICE Time (s)	30.24	59.97	60.52	62.38	66.18	211.17
FIRE Time (s)	0.61	1.01	1.46	4.12	6.03	135.82
Speedup (factor)	**50**	**60**	**41**	**15**	**11**	**2**

of the spherical harmonics field is increased. The final state vector of FIRE is off by 15 meters (approximately) when compared to the final state vector obtained by using SPICE for a typical 100 day trajectory. This is well within acceptable limits considering the spacecraft trajectory is highly perturbed and it makes hundreds of revolutions (approximately 1,336) around the Earth during the integration.

Figures 69 and 70 show the evolution of the orbital elements (for 20 days) and the resulting trajectory for the fully perturbed plus 10 by 10 gravity field case. Figure 71 gives the instantaneous position differences between FIRE and SPICE propagations. We note that the orbital elements are reported in the body-fixed frame at epoch.

4.5.2.2   Saturn-Moons-Jupiter-Sun (SMJS)

Here we consider the case of Saturn, its nine moons, Jupiter and the Sun (SMJS). A highly elliptical orbit with an apoapsis of 1,518,900 km (near Titan) and periapsis of 141,100 km (close to Mimas) around Saturn is selected for this propagation. This orbit enables us to capture the gravity signature of all the nine moons of Saturn, the Sun, and Jupiter. The trajectory propagation is carried out for 200 days. We only consider an 8 by 0 gravity field for Saturn. The initial conditions are given in Tables 23 and 24 and the results are given in Table 25. For the typical case, FIRE's

114
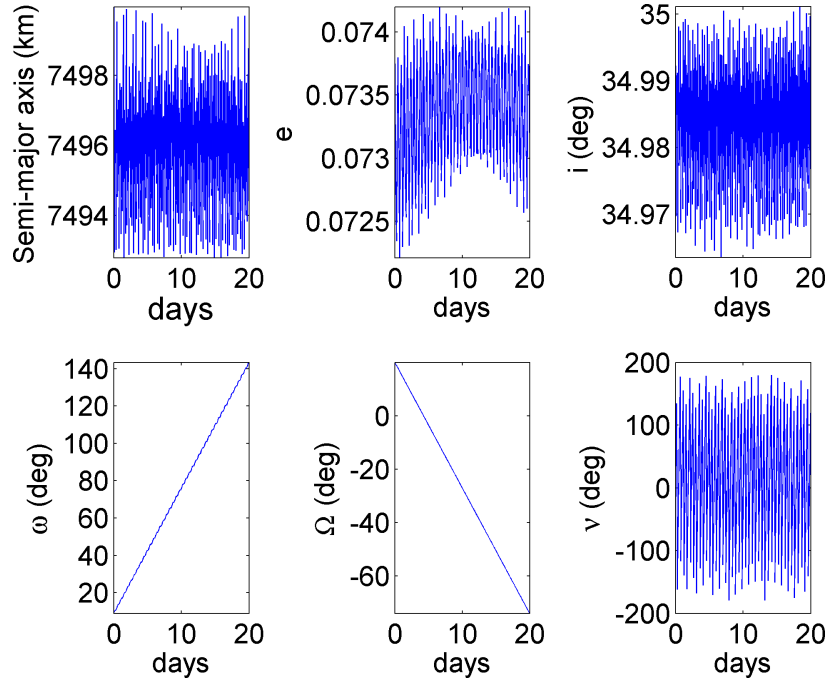
Figure 69: Evolution of orbital elements for EMS system

Table 23: Initial condition (body-fixed frame at epoch) for SMJS trajectory

Orbital Parameter	Value
Semi-major axis (a)	830000 (km)
Eccentricity (e)	0.83
Inclination (i)	9 (deg)
Argument of periapsis (ω)	2 (deg)
Longitude of ascending node (Ω)	11 (deg)
True anomaly at epoch (ν)	0 (deg)
Epoch (t0)	2454477.50 (JD)

Table 24: Gravity field at Saturn

Case	Acceleration combination
1	two body
2	2 by 0 (J2 only) + two body
3	8 by 0 (J2-J8) + two body

trajectory integration ran 70 times faster. The final state vector is off by 21 meters (approximately) which again is well within acceptable limits given the long flight time (approximately 22 revolutions) and highly perturbed and eccentric nature of the trajectory.

Figure 72 shows the evolution of orbital elements for the perturbations plus J2 acceleration (2 by 0) case. Amongst Saturn's moons, Titan is the largest and its gravity signature appears as jumps in the orbital elements. The orbital elements are shown for the first 100 days only. Figure 73 shows the resulting highly eccentric trajectory propagation. Figure 74 gives the instantaneous position differences between
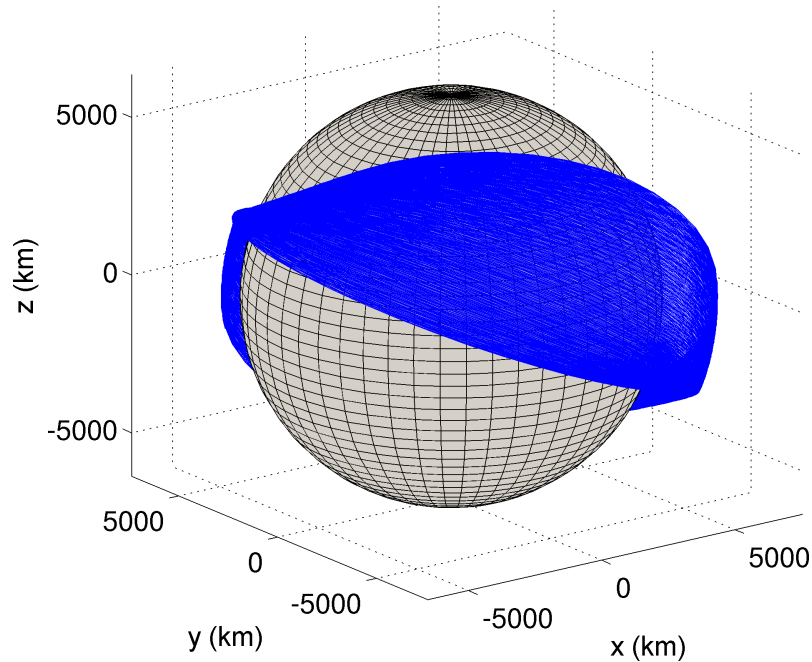
Figure 70: Propagated trajectory in EMS system

Table 25: Performance comparison ratio table for SMJS trajectory propagation

Cases →	Case 1	Case 2	Case 3
SPICE Time (s)	61.79	81.07	81.79
FIRE Time (s)	1.06	1.16	1.36
Speedup (factor)	**59**	**70**	**60**

the FIRE and SPICE propagations.

4.6   Chapter Conclusion

In this chapter a fast, efficient, smooth, and accurate ephemeris interpolation system called FIRE is proposed for general use in precision trajectory and mission design. The FIRE system is custom built for applications traditionally bogged down by the heavy computational burden associated with typical ephemeris calls. FIRE relies on established ephemeris systems (like JPL's SPICE) to build a custom archived ephemeris, hence FIRE is intended as a supplemental capability for users that benefit from fast calls and smooth derivatives. The main disadvantage of FIRE is the added layer of complexity and the additional (though modest) memory requirements during runtime.

116
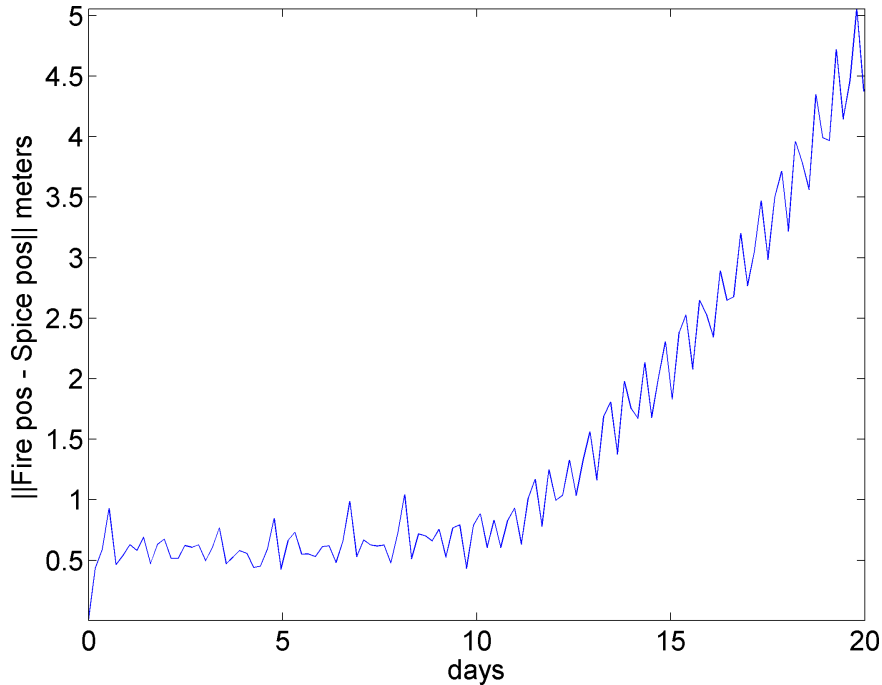
Figure 71: SPICE and FIRE position difference (EMS)

We demonstrate speed improvement of approximately one order of magnitude for typical applications when compared to similar functionality in SPICE, one of the most widely used ephemeris system. The main performance gain is achieved through the modest use of random access memory to store a custom, portable runtime ephemeris. A major benefit of FIRE is that smooth, accurate, and self-consistent derivatives are natural artefacts of the interpolation method. These derivatives are often required for trajectory optimization and other classes of similar problems. At the time of writing this thesis the FIRE software has already undergone extensive testing and will be released to the general public via the Internet.

FIRE has been designed specifically to facilitate problems involving long or frequent ephemeris propagations for various classes of orbital mechanics problems. The new tool has potential value to any high precision application or software requiring fast, accurate, and smooth ephemeris data. Chapter 6 demonstrates the use of FIRE along with other perturbation model for performing high-fidelity multiple spacecraft
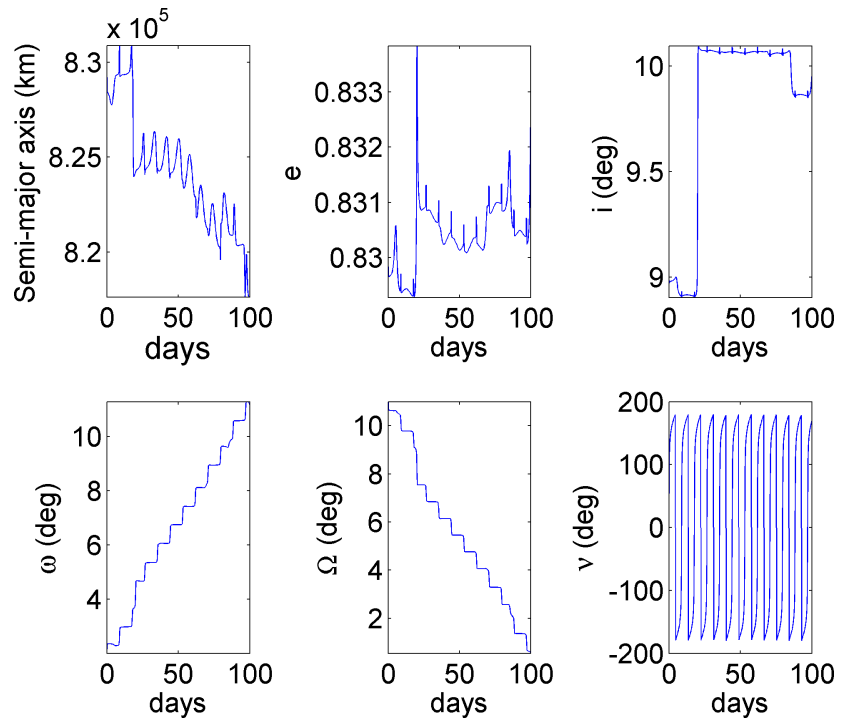
Figure 72: Evolution of orbital elements for SMJS system simulations.
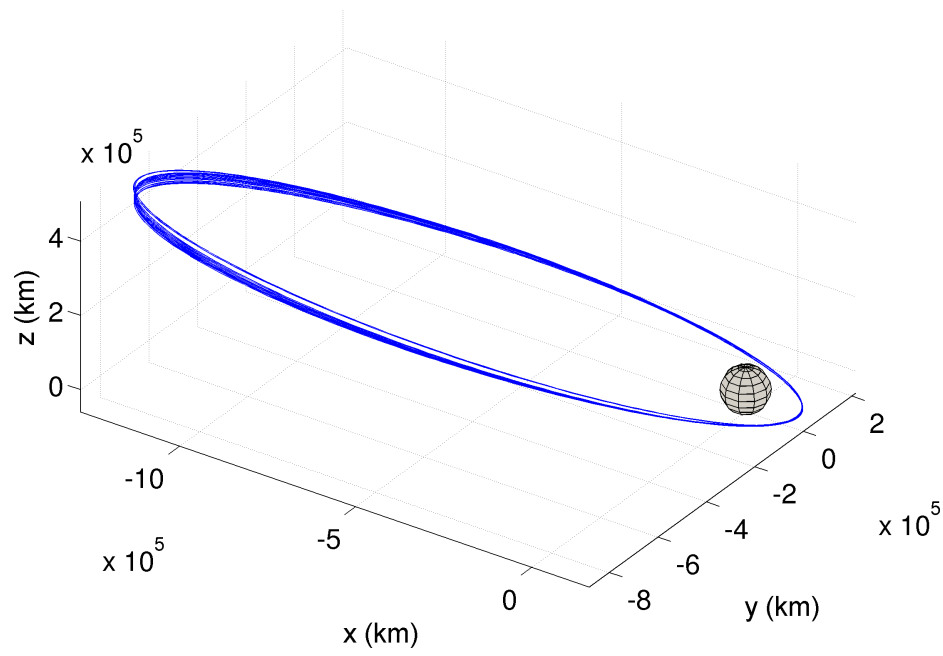
Figure 73: Propagated trajectory in SMJS system
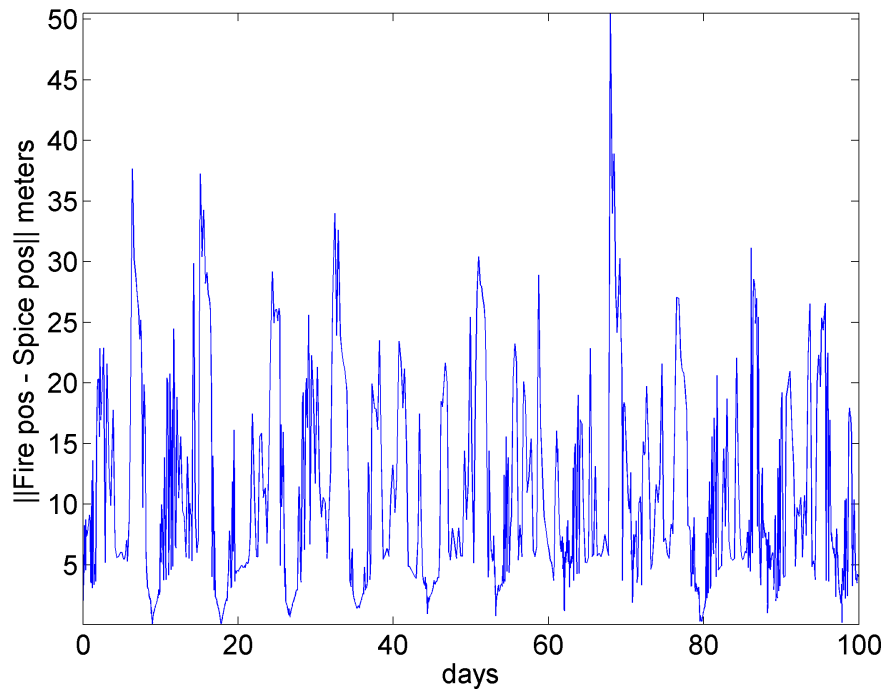
Figure 74: SPICE and FIRE position difference (SMJS)

CHAPTER V

FAST AND ACCURATE SENSITIVITY COMPUTATIONS

5.1 Chapter Summary

Gradient based trajectory optimization relies on accurate sensitivity information to robustly move a solution towards an optimum. Computational complexity of sensitivity calculations increases exponentially for higher problem dimensions and orders. Hence, the computation of these sensitivities is traditionally a major speed bottleneck in trajectory optimization and targeting algorithms. In this chapter a novel methodology using NVIDIA's GPU (Graphics Processing Unit) to rapidly calculate the derivatives in a multilevel, parallel, and heterogeneous way while the CPU (Central Processing Unit) sequentially computes the less expensive state equations, is proposed. A tool, based on the methodology, computes both the first and second order analytic sensitivities on the GPU with double precision accuracy. For an example trajectory propagation, overlapped computations are demonstrated such that first order sensitivities are calculated almost for free compared to the conventional CPU implementation.

5.2 Chapter Nomenclature

t	Time
y	State vector
f	Equations of motion for the state
g	Inequality constraint vector
c	Equality constraint vector
X	Nominal state vector
I	Identity matrix
J	Performance index or cost
n	Dimension of state vector
$[x, y, z]$	Position vector

$[u, v, w]$	Velocity vector
G	Standard gravitational parameter
M	Mass of the body
ϕ^1	First order state transition matrix
ϕ^2	Second order state transition tensor
N_s	Number of sub-trajectories
N_t	Total number of integration steps
SP	Single precision
DP	Double precision
TR	Thread recursion
STM	State transition matrix, $\in \Re_{N \times N}$
STT	State transition tensor, $\in \Re_{N \times N \times N}$
GPU	Graphics Processing Unit
CPU	Central Processing Unit
ODE	Ordinary differential equation
$Gflop/s$	Giga floating point operations per second
$nsub$	Total number of points per sub-trajectory (multiple of 16)
$scale$	Scaling parameter for nsub
$s/c\ revs$	Space craft revolutions
$CUDA$	Compute Unified Device Architecture
$NVCC$	NVIDIA C compiler
$GPGPU$	General-purpose computing on graphics processing units

Conventions

i	ith sub-trajectory
j	jth point on a sub-trajectory
T	Transpose
δx	Very small change in x
$\dot{x}$	Complete derivative of x with respect to time
$A*B$	Matrix times tensor, A is a matrix and B is a tensor: $A*B = \Sigma_k\ A(:,:)B(:,:,k)$
A^T*B*A	Matrix transpose times tensor times matrix,
	A is a matrix and B is a tensor: $A^T*B*A = \Sigma_k[A(:,:)^T B(k,:,:)A(:,:)]$

5.3   Introduction and Background

Gradient based numerical optimization is used in used in all areas of science and engineering.[117,142,40,119] Various sub-fields in numerical optimization such as Optimal Control[22,110,109,123,104] and Parameter Optimization,[60,47] and other gradient based

continuous methods make use of numerical sensitivities to select new step directions.

Most trajectory optimization algorithms make use of higher order sensitivity information[100, 18, 38, 95] to robustly move an optimum. Computational burden of sensitivity calculations scales exponentially as a function of problem complexity and order of the derivatives. Given a function evaluation computational complexity of $O(n)$, the corresponding first order sensitivities have a computational complexity of $O(n^2)$, and similarly second order sensitivities have a computational complexity of $O(n^3)$. The current state of methodology relies on serial computation of sensitivities on the CPU, and is often not feasible to solve realistic model problems because of the extraordinarily expensive sensitivity calculations. This complexity (and the high costs of large CPU clusters required to overcome) therefore prohibits many classes of high-fidelity optimization problems from being solved.

Previous work in parallel sensitivity analysis has been limited to a small class of problems.[21, 24, 20, 29, 51] These methods generally fail and exploit the massive parallelism present in the underlying problem. Furthermore, most of the previous researchers have focused on the problem of parallelizing sensitivity computations either across a multidimensional solution domain[136] or to multiple differential-algebraic equations (DAE).[146, 51] In the thesis we intend to solve the problem efficiently parallelizing sensitivity computation across a single solution trajectory (or state integration).

The NVIDIA's CUDA (Compute Unified Device Architecture[91]) technology enables an innovative solution to the above problem. NVIDIA's GPU architecture is tailor made to exploit fine grain parallelism and CUDA makes it possible to program the GPU hardware efficiently. With the introduction of double precision capability, it is now possible to achieve dramatic speedups (without loosing accuracy) if new and innovative algorithms utilize the large amount of parallelism efficiently.

In this chapter a new tool is proposed, which exploits heterogeneous programming by utilizing the CPU and NVIDIA's Graphics Card (GPU) together to achieve

substantial speedups for sensitivity computation. The proposed algorithm breaks the CPU derived solution trajectory (or solution path) into numerous smaller blocks and solves the associated sensitivities in a heterogeneous parallel manner on the GPU. These multiple levels of parallelism exploit the fine grained architecture of the GPU, resulting in significant performance gains. A similar decoupling of the state and sensitivity computation in done in Ref.[51] but applied to problem of integrating multiple DAE's. On the other hand, the methodology proposed in this chapter attempts to expose and exploit the multi-level parallelism in a single state integration.

Comparison with a CPU only simulation on an example Keplerian trajectory is performed. The speedup for a two body trajectory plus sensitivity propagation over the complete CPU implementation is ∼4 times and ∼14 times for first order STM and second order STT sensitivity evaluations, respectively. The tool is general in its design and implementation subject only to user defined equations of motion. The fast sensitivity propagations can therefore be useful to a wide variety of gradient based optimization or targeting problems.

In the forthcoming sections of this chapter the general sensitivity formulation problem is defined, next a brief introduction of the NVIDIA GPU and the CUDA programming model is provided which is followed by brief overview of the algorithmic implementation, and finally the performance results are presented.

5.4   General Sensitivity Formulation

Numerical optimization refers to maximizing or minimizing a continuous function subject to certain constraints and input variables. A general numerical optimization strategy is shown in Fig 75. The black box function can be any continuous function of the input.

A common and general optimization problem involving state equations is mathematically defined as follows:
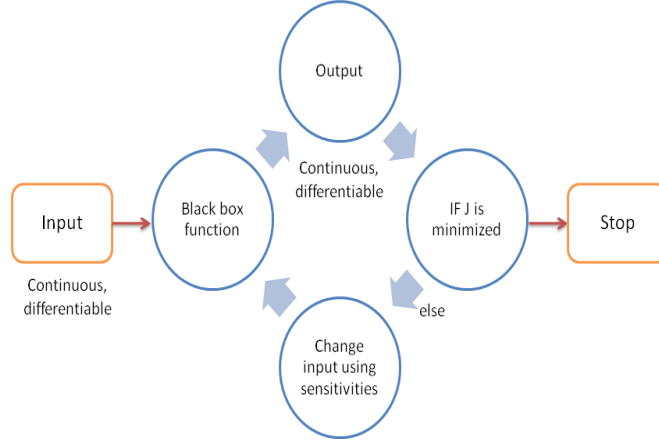
Figure 75: General Solution Strategy

$$\min_{y(t_0)} J(y_f, t_f) \ , \ \text{ subject to } \begin{cases} \dot{y} = f(y, t) \\[2mm] c(y_f, t_f) = 0 \\[2mm] g(y_f, t_f) < 0 \\[2mm] y \in \Re, t \in \Re + \end{cases}$$

Here J is the performance index which we want to minimize, y is the vector of state (and control) variables, t is time, f(y) represents dynamics of the system, $c(y)$ and $g(y)$ are the equality and inequality constraints (of arbitrary dimension) on the state vector. If $J=0$ the problem reduces to a targeting or boundary value problem.

To solve the above problem using gradient methods, the sensitivities (derivatives) of the final state vector with respect to the initial state vector are required. The first order derivatives can be computed using numerical differencing of the function or analytically by direct integration of the so-called state transition matrix (STM).[110, 13] Many solution techniques (Newtons method for example) require second order derivatives to guide the solution efficiently towards the local optimum. The second order state transition tensors (STT) can also be calculated via numerical differencing or direct integration.[100, 18, 38, 95] The STM and STT are used to map derivatives from one time to another on a given continuous trajectory. Please refer to[13, 122, 95] for

detailed discussion of STM and STT. The STTs have an exceptionally high computational cost associated with them. Hence, second order derivatives (Hessians) are usually only approximated; and full second order derivatives are only used in specialized high-fidelity methods.[79, 97] Many important trajectory optimization problems are highly non-linear in nature making these higher order sensitivity computations very attractive in the solution process.

The general Taylor series expression for the first order STM and the second order STT about the nominal state (X) is given by Eq. 90

$$\delta X_{j+1} = \phi^1 \delta X_j + \frac{1}{2} \delta X_j^T * \phi^2 * \delta X_j \tag{90}$$

These highly coupled sensitivities (ϕ^1 and ϕ^2) are evaluated alongside the integration of the state vector by solving Eq. 91 and Eq. 92.

$$\dot{\phi^1} = f_x \phi^1 \tag{91}$$

$$\dot{\phi^2} = f_x * \phi^2 + (\phi^1)^T * f_{xx} * \phi^1 \tag{92}$$

subject to initial condition $\phi^1(t_o) = I_{n \times n}$ and $\phi^2(t_o) = 0_{n \times n \times n}$

The complexity of computing the sensitivities in terms of flops (floating point operations per second) is of the order $O(n^{p+1})$, where n is the dimension of the state vector and p is the order of sensitivity required. In this study we consider only up to $p = 2$. Consider a typical trajectory problem of dimension 6. The STT and STM are of dimension 6×6 and $6 \times 6 \times 6$, respectively. A concurrent evaluation of the state, STM, and STT therefore requires numerical integration of 6+36+216 coupled equations. Note that the STT dimension can be reduced to $n(n + 1)/2$ if symmetry is considered.

Although the successive steps of the state trajectory must be computed in sequential form, the STM and STT can be calculated in parallel once all points of the state are known. We build upon this insight and use the NVIDIA GPU hardware

with the help of CUDA technology to achieve substantial performance improvement. Appendix D gives brief overview of the NVIDIA GPU architecture and the CUDA programming language.

The next section highlights the implementation of the fast sensitivity calculation tool.

5.5 Heterogeneous Sensitivity Computation

At any given point on the solution trajectory, the STMs and STTs (of any order) are a function of the state vector and time at that point. Hence, these sensitivities can be evaluated in parallel once we obtain the state information for the whole trajectory.

Given two STMs which map the partial derivatives between times t_i to t_{i+1} and between times t_{i+1} to t_{i+2}, then the equivalent STM mapping between times t_i to t_{i+2} is given by the chain rule in Eq 93:

$$\phi^1(t_{i+2}, t_i) = \phi^1(t_{i+2}, t_{i+1})\phi^1(t_{i+1}, t_i) \tag{93}$$

For a second order STT, the mapping expression from one time to another is more complicated and is calculated in Eq 94.

$$\phi^2(i+2, i) = \phi^1(t_{i+2}, t_{i+1}) * \phi^2(t_{i+1}, t_i) + (\phi^1(t_{i+1}, t_i))^T * \phi^2(t_{i+2}, t_{i+1}) * \phi^1(t_{i+1}, t_i) \tag{94}$$

Herein lies the motivation to compute STMs and STTs in parallel. Given N_t number of integration steps required for the state trajectory, the STM and STT from point i to $i+1$ for all $i = 1..N_t - 1$ can be calculated in parallel with initial conditions I and 0 respectively. The final state sensitivities with respect to the initial state are then calculated with the recursive evaluations of Eq. 93 and 94. We call this final step the reduction or reduce step.

The next section discusses this solution strategy in detail.
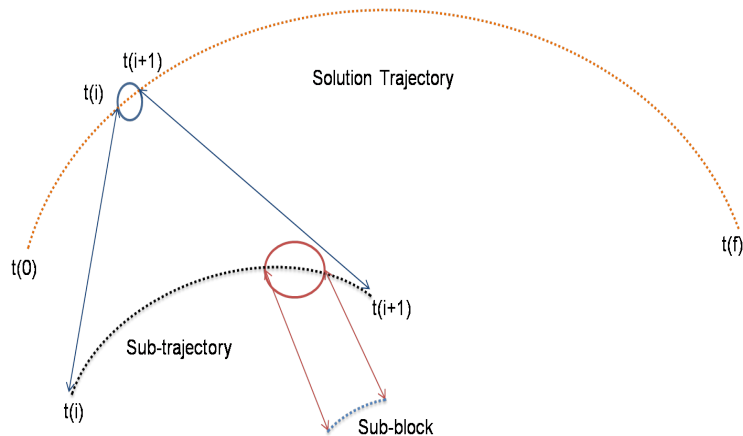
Figure 76: Solution Strategy

5.5.1 Solution strategy

We start by breaking the CPU generated sequential trajectory into multiple sub-trajectories with each sub-trajectory consisting of $nsub$ number of integration steps. Figure 76 shows the basic solution strategy which further breaks a sub-trajectory into various blocks, with each block containing a certain number of points. As only state information is required to compute the sensitivities between two points in a particular block, this structure is perfectly suited for explicit parallelism. The sensitivities within each block are mapped to CUDA thread blocks and multiple blocks are joined together to form a CUDA grid block. The whole grid block is then evaluated in parallel. The final sensitivity matrix is calculated via the chain reduction on the CPU. Further, these computations are repeated for each sub-trajectory which gives rise to multiple levels of parallelism and permits concurrent execution.

To calculate the sensitivities a GPU kernel is invoked as soon as we have the CPU integrated state at each point on a sub-trajectory. So while the GPU is evaluating the sensitivities for the sub-trajectory (i) the CPU advances with the state integration for the sub-trajectory ($i+1$). This enables overlapping the GPU sensitivity computation with the CPU state integration. Typically, for our first order STM computation the GPU finishes before the CPU has finished integrating the next sub-trajectory.

This results in an almost complete computation overlap between the two hardware's, except for the last sub-trajectory computation on the GPU. This heterogeneous computation strategy along with an intelligent memory copy operation exploits the GPU architecture efficiently. The basic execution strategy is the same for both the first and second order STM and STT evaluation.

Next we elaborate on the specifics of the first and second order implementation.

5.5.2   First order STM implementation

The evaluation of the first order STM from one point to the next is divided into 4 kernel calls. The first kernel is responsible for calculating the initial function evaluation and initializing the global memory for each thread. The global memory holds the state information and the step size taken by the integrator at each point. The next two kernels execute 12 times (sequentially) corresponding to the 12 function evaluations required per time step in the DOPRI-78, Dormand Prince integrator[36] (implemented on the GPU). After execution of kernels 1,2 and 3 we obtain the final state transition matrices between subsequent points on the current sub-trajectory being evaluated.

The kernel 4 is then invoked to locally reduce the STMs in each thread block to a single STM. This operation uses a thread recursion (TR) algorithm. To facilitate faster parallel computations, threads of a particular thread block load matrices in a specific order, similar to parallel reduction algorithms on the CPU. The TR algorithm for a thread block size of 8 is given in appendix A (algorithm 2). The TR algorithm is shown in appendix A is for 8 threads per thread block, we use 256 threads per thread block in our actual implementation.

After kernel 4 is finished we are left with limited number of state transition matrices which have to be multiplied (in decreasing order) to get the final state transition matrix for the sub-trajectory. The above procedure except for the final CPU reduction

step is repeated for various sub-trajectories. The final reduction step is performed all together for each sub-trajectory on the CPU after the state integration.

Using the TR algorithm we are able to significantly reduce to the number of matrices which multiply in the final reduction step on the CPU. This strategy also imparts numerical stability to the STM evaluation as we are always multiplying matrices which are of the same order (approximately).

Figure 77 shows the TR algorithm and the final CPU reduction operation.
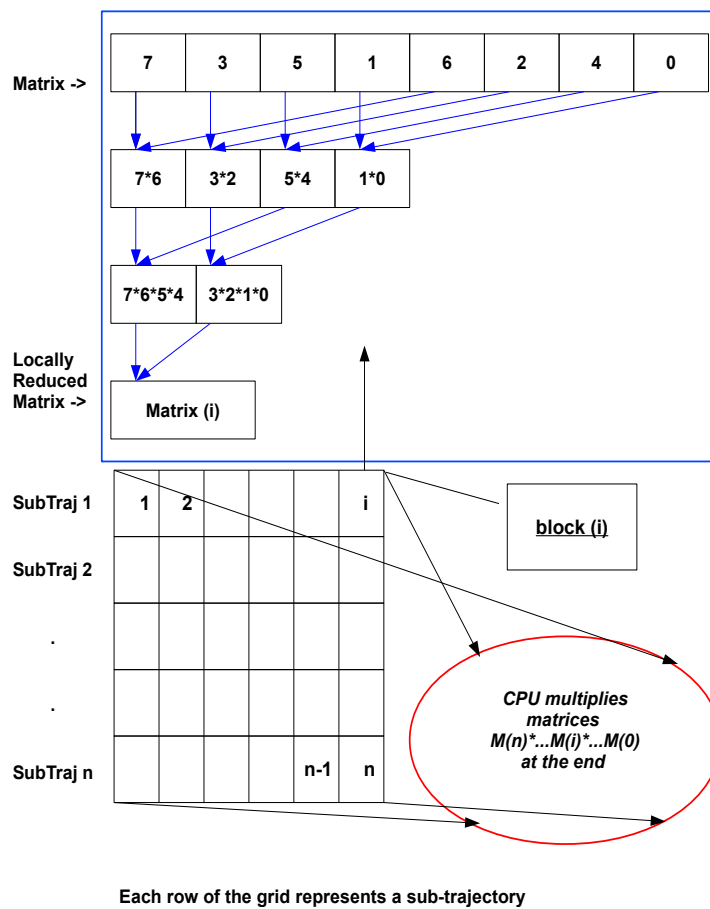
Figure 77: TR algorithm (graphical) and final CPU reduction

5.5.3 First order STM + Second order STT implementation

The second order implementation for each sub-trajectory is accomplished by 6 kernel calls and one final complete CPU reduction. As in the previous case, kernel 1 performs the initial function evaluation and global memory initialization for each thread. Kernels 2 to 6 are called 12 times sequentially, each performing a part of a single function evaluation for one step of the integrator. Specifically, kernels 4 and 5 carry out required the matrix tensor products needed for the second order STT integration. After all kernels are finished we obtain the full first order STM and second order STT between subsequent points on the sub-trajectory. This operation is repeated for all the sub-trajectories, followed by a complete reduction to obtain the final sensitivities on the CPU.

5.5.4 User interface

We basically replicate the capability of a general integrator where the user provides a set of routines which perform the function evaluation. These routines are programmed in C programming language. The user has full control over the parameters which directly affect the performance of the tool, like the number of points in a sub-trajectory ($nsub$), number of thread blocks, number of sub-trajectories (N_s), etc.

The user routines (both for first order STM and second order STT) should be optimized for minimizing global memory transfers and avoiding shared bank conflicts (avoiding threads to read from shared memory in a random fashion) even at the expense of doing more floating point operations (flops). This is often the case for a GPU kernel, as global memory operations are generally more expensive (up-to 300 times slower) then floating point operations on the GPU.

By default the number of points per sub-trajectory ($nsub$) is set to $30,720/scale$. The parameter $nsub$ has to be a multiple of 32 to achieve high global memory performance on the GPU. By default, the scale parameter has a value of 4 for the first

order STM computation and 2 for the second order STT computation. The code automatically handles the last sub-trajectory branch evaluation by launching empty threads on the GPU, if N_t (number of steps taken by the CPU integrator) for the complete trajectory is not a multiple of $nsub$.

Figure 78 depicts the general heterogeneous algorithm.
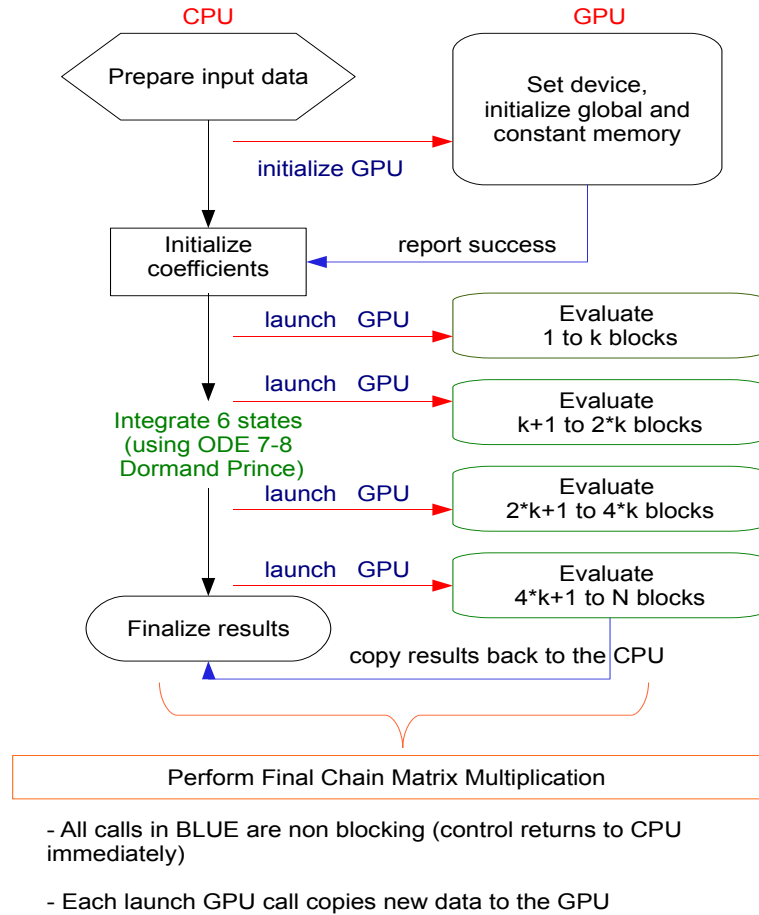
Figure 78: General hetrogenous algorithm for sensitivity computation

5.6   *Results*

In this section we evaluate the performance of our new tool against an optimized CPU implementation. A 2-body near earth propagation is used as the test trajectory. We evaluate performance for both the first order STM and second order STT

implementation. For the 2-body case the order of integration of the state vector (y) is 6. Hence the order of integration for the STM computation is 42 (36+6) and the order for the STM plus STT computation its 258 (6 + 36 + 216). We are aware of the symmetry present in the second order STT computation but we currently choose to avoid the added complication in the GPU implementation.

Table 26 states the initial conditions for the propagated trajectory.

Table 26: Initial condition (body-fixed frame) for trajectory integration

Orbital Parameter	Value
Semi-major axis (a)	8300 (km)
Eccentricity (e)	0.49
Inclination (i)	35 (degrees)
Argument of periapsis (ω)	9 (degrees)
Longitude of ascending node (Ω)	20 (degrees)
True anomaly at epoch (ν)	0 (degrees)

For the current computation, the *scale* parameter is set to a default value of 4 for the first order STM computation and 2 for the second order STT computation for all the results. Hence, the number of points per sub-trajectory is 7,680 and 15,360, respectively. The *scale* parameter is a function of both the GPU hardware and complexity of the problem being solved.

Table 27: Test hardware specifications

Component type	Component
CPU	Intel Core 2 Duo E6550 @ 2.33 Ghz
Operating system	Linux X86_64
GPU	Tesla C1060
Memory	4 GB

Table 28: Maximum theoretical performance comparison

Criteria	CPU	GPU
Max SP Gflop/s	24	933
Max DP Gflop/s	12	78

5.6.1 Test Hardware

Table 27 gives the specifications of the test hardware. Table 28 gives the theoretical performance of the CPU and GPU used for this example. The CPU code is compiled with the Intel Fortran compiler version 11.0 with optimization level set to "*-fast*". This enables auto vectorization and inter-procedural optimization. These optimizations result in a 2 times improvement in performance over the un-optimized CPU code. Apart from compiler optimization the CPU code is tuned for high performance Fortran. The CUDA code is compiled using the NVCC compiler version 3.0 . All computations are carried out using a RK-78 Dormand Prince integrator[36] set to normalized tolerance of 1E-14. For consistency and importance to the astrodynamics community, IEEE compliant double precision arithmetic has been used for all the results presented.

Table 29: Performance table (time, sec) for first order sensitivity computation

Tof (days)	State only (CPU)	State + STM (CPU)	State + STM (GPU plus CPU)
4.25	0.10	0.38	0.12
17.00	0.39	1.52	0.41
25.00	0.58	2.29	0.60
68.00	1.54	6.11	1.58
100.00	2.32	9.15	2.35
136.00	3.09	12.19	3.13
200.00	4.54	17.86	4.60

5.6.2 First order STM computation

Table 29 gives the performance of our tool compared to the corresponding CPU implementation for first order STM plus state computation.

We define speedup by Eq. 95

$$speedup = \frac{(CPU\ time\ for\ integrating\ sensitivities\ along\ with\ the\ state)}{(CPU\ time\ for\ integrating\ the\ state + GPU\ time\ for\ integrating\ sensitivities)} \quad (95)$$

This speedup is always less then the theoretical maximum speedup, defined by Eq. 96

$$speedup_{max} = \frac{(CPU\ time\ for\ integrating\ sensitivities\ along\ with\ the\ state)}{(CPU\ time\ for\ integrating\ only\ the\ state)} \quad (96)$$
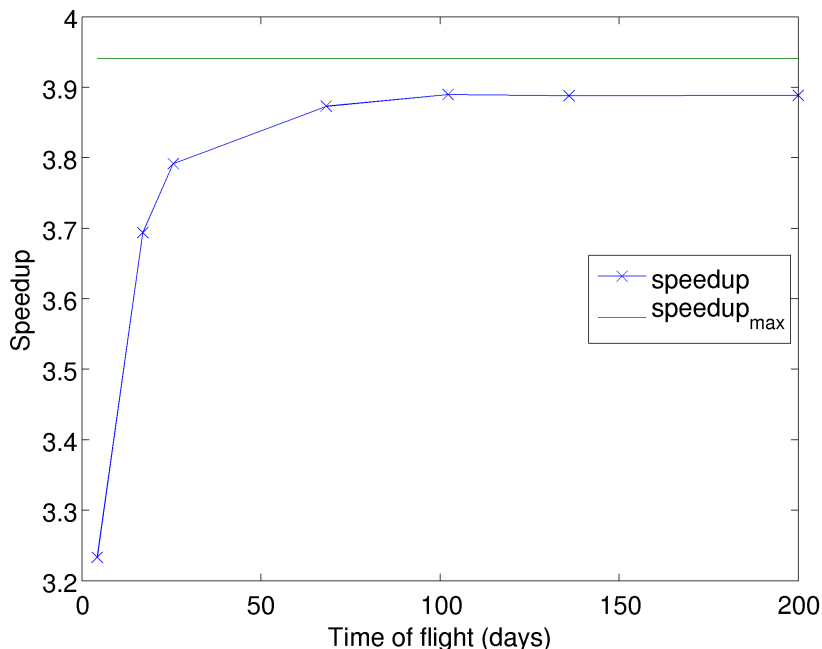
Figure 79: Speedup for complete STM computation

Figure 79 shows the speedup we achieve over the full CPU implementation with respect to time of flight. As we are able to almost completely hide the first order STM calculations on the GPU, the final speedup approaches the theoretical maximum value (Fig 79). In terms of speed, our GPU implementation of the STM plus state is almost as fast as the the CPU implementation of the states only. Therefore, we approximately achieve the conventionally difficult to compute STM calculations almost for free. We further note that, for a computationally more expensive STM calculation (e.g higher dimension state and complicated force models) the theoretical speedup limit will be much higher and so will be our speedup over the CPU implementation.

5.6.3 First order STM + Second order STT computation

Table 30 gives the performance of our tool compared to the corresponding CPU implementation for the second order STT, the first order STM, and state computation.

Figure 80 shows the speedup over the CPU implementation with increasing time of flight. We can see that the speedup is more impressive than the first order STM

Table 30: Performance table (time, sec) for first order sensitivity computation

Tof (days)	State only (CPU)	State + STM + STT (CPU)	State + STM + STT (GPU plus CPU)
4.25	0.10	3.58	0.32
17.00	0.39	14.29	1.15
25.00	0.59	21.63	1.76
68.00	1.53	56.10	4.54
100.00	2.31	85.00	6.77
136.00	3.06	112.50	8.43
200.00	4.48	167.70	11.94

implementation because the dimension of the STT is 6 times larger. Due to the final complete reduction being done on the CPU (as opposed to the first order STM case) we are not able to efficiently overlap the computations between the GPU and CPU. Still we are able to achieve an order of magnitude speed improvement over the CPU implementation. As the GPU favors computation over memory operation, we expect this speedup value to be higher for more computationally expensive STT evaluations.

It is worthwhile to note that the GPU performs 8 times (approximately) faster in single precision mode than in double precision mode.

5.6.4 Numerical accuracy

The ODE 78 integration on the GPU is accurate up to 13 digits when compared to the CPU integration. This has been achieved by designing numerically stable algorithms and by using precision preserving math functions on the GPU. The final first order STM and second order STT have relative errors of 1E-13 (approximately) for smaller propagations ($>=$100 s/c revs). For longer propagations ($>=$1000 s/c revs) the final difference in the computation on CPU and GPU is 1E-11 (approximately).

It is well known that after large number of space craft revolutions both the STM and STT become very large. On the CPU computing these sensitivities leads to rounding errors as large matrices are multiplied by small matrices at each successful integration step. While on the GPU, as the matrices are reduced in thread blocks in
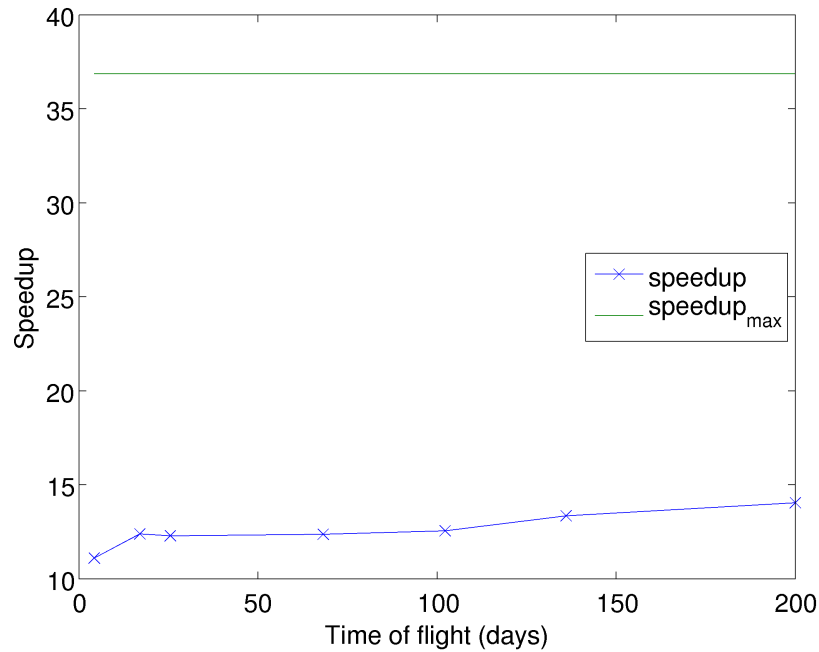
Figure 80: Speedup for complete STM plus STT computation

parallel, they are always of approximately the same order during multiplication. Even when they are finally reduced on the CPU they have a more stable rounding error behavior, as the matrices are again of the same order approximately. This explains the increase in relative difference as the number of revolutions of the spacecraft increases. The STM from the GPU are therefore closer to the truth than those computed on the CPU for long propagations.

5.7   Chapter Conclusion

A new tool is proposed in this chapter, capable of computing first and second order sensitivities in parallel for gradient based numerical optimization. The proposed tool implements a heterogeneous algorithm, utilizing both the CPU and GPU concurrently to achieve substantial performance increase. The tool is able to compute the first order sensitivities for almost no extra computational cost than compared to integrating just the states on the CPU. The second order sensitivities are computed at up to 14 times faster for example 2-body trajectory integration. The difference

in the calculated sensitivities when compared with their CPU counterparts is 1E-13, approximately. Further, the implementation is numerically more stable for long trajectory propagations compared to an equivalent CPU implementation. The tool is general in its design and can applied to any gradient based optimization problem which requires fast and accurate sensitivities.

Given the performance, accuracy and generality of the proposed solution methodology (and the developed tool), it is well suited for a wide range of numerical optimization problems. Problems which are intractable due to their high computational complexity and/or dimension may now be attempted more readily without the burden of slow sensitivity calculations.

CHAPTER VI

MULTIPLE SPACECRAFT TRAJECTORIES USING GPU COMPUTING AND FAST, HIGH-FIDELITY GRAVITY PERTURBATION MODELS

6.1   Chapter Summary

To achieve both speed and accuracy in multi-spacecraft trajectory simulations, a solution methodology is presented that takes advantage of 1) the Fetch and FIRE, high-fidelity geopotential and third-body perturbation models, respectively, that efficiently trade memory for speed and 2) a Graphics Processing Unit (GPU) based integrator to achieve parallelism across multiple spacecraft. The two methods combined lead to multiplicative speedups, making the tool almost five orders in magnitude faster, in some extreme cases, compared to the same simulation performed in serial on a single CPU. For state-of-the art space-catalog applications the tool is found to be two to three orders in magnitude faster. The solution approach is highly relevant to the conjunction problem, covariance realism, particle filters and Monte-Carlo analyses.

6.2   Chapter Nomenclature

$a, e, i, \omega, \Omega, \nu$	Classical orbital elements
x, y, z	Components of spacecraft position vector
u, v, w	Components of spacecraft velocity vector
$\vec{r}$	General position vector
c	Speed of light
m	Mass of a spacecraft
A	Projected area of a spacecraft
h_0	Base altitude
H	Scale height
ρ_0	Nominal density
r_p	Periapsis distance

r_a	Apoapsis distance
C_r	Coefficient of reflectivity
C_d	Coefficient of drag
Φ	Solar intensity
$\vec{a}_{tot}$	Total acceleration
$\vec{a}_{eph}$	Lunisolar (ephemeris) acceleration
$\vec{a}_{grav}$	Higher order gravity field acceleration
$\vec{a}_{drag}$	Acceleration due to atmospheric drag
$\vec{a}_{srp}$	Acceleration due to solar radiation pressure
G	Standard gravitational parameter
GPU	Graphics Processing unit
LEO	Low Earth Orbit
MEO	Medium Earth Orbit
RAM	Random Access Memory
FIRE	Fast Interpolated Runtime Ephemeris
SPICE	Spacecraft Planet Instrument C-matrix Events

6.3 Introduction and Background

The increasing number of objects in space and the increasing complexity of space missions coupled with improving surveillance accuracy is driving the need for high-fidelity spacecraft trajectory design tools. A variety of space surveillance applications, such as real time tracking, the conjunction problem, particle filters and orbital debris tracking are currently slowed down by state-of-the art integration methods and force models. The speed bottleneck associated with trajectory integration has be studies by several authors.[61,10,44] Both sinlge-step and multi-step techniques have been used for trajectory integration. Recent alternatives like collocation methods[10] and Taylor series[120] based methods are actively being researched. Numerous past studies have also focused on increasing the performance of the force calculations.[71,106,112,88,68,111,8,6,19] Often, truncated[16,26] and semi analytic techniques[103,61] are adopted, which can compromise the accuracy.

Similar to the previous chapter, the massive parallelism present in the current

problem is well-suited for the new NVIDIA Graphics Processing Unit (GPU) archi-tecture. The new GPU architecture enables advanced features like zero-copy which is essential for complex problems with large volumes of solution data. The advan-tage of GPU based parallelism lies in its single user capability without the need for expensive CPU clusters. The attractive compute speeds of the semi-analytic models can be realized on the GPU without sacrificing accuracy.

In this chapter we present a methodology which enables large scale, high-fidelity integration of multiple spacecraft. To achieve this goal we bring together the fast and accurate perturbation models (the FIRE and Fetch models developed in chap-ters 3 and 4) on the GPU, together with a GPU based Runge-Kutta integrator to achieve massive parallelism across multiple spacecraft. The previously developed FIRE ephemeris model was shown to be capable of providing up to two orders of magnitude in speedup over the widely used JPL's SPICE model[2] and the Fetch grav-ity model, which was demonstrated to provide up to three orders of magnitude in speedup over a state-of-the-art non-singular spherical harmonics approach. These perturbation models combined with a GPU based Runge-Kutta solver lead to mul-tiplicative speedups, when compared to a serial single CPU implementation. The performance of this new tool is evaluated in two configuration cases: 1) objects that are in close proximity 2) objects that are randomly distributed in the LEO/MEO en-vironment. The integration algorithm utilizes advanced GPU programming features like zero-copy for data transfer and is only limited by the amount of Random Access Memory (RAM) available. Results are presented for a grid including simulation flight times of up to 2.5 days and number of objects up to 100,000. The methodology, im-plemented on the NVIDIA M2090 GPU, is found to be five orders in magnitude faster in some extreme cases, compared to the same simulation performed in serial on a sin-gle CPU. The solution approach presented in this chapter is highly relevant to the conjunction problem, covariance realism, particle filters, and Monte-Carlo analyses.

It is emphasized that the current best practices do not always warrant the use of a full 360×360 geopotential due to the high level of error sources in other terms such as drag and solar pressure radiation. Furthermore, current best practices may use analytic approximations of the third body perturbations, which are much faster to compute than perturbations via SPICE. Therefore, the speedups presented in this chapter are not expected to be fully realizable compared to current state of the art techniques, but rather compared to the equivalent serial computations using SPICE and the high degree and order spherical harmonics.

The next section of this chapter discusses the general computational structure and adopted solution strategy. Note that the terms "object" and "spacecraft" are equivalent for the current study.

6.4 General Computational Structure

Following a parallel heterogeneous approach similar to that in chapter 5, the proposed solution strategy takes advantage of both the CPU and GPU working in tandem to achieve multiplicative speedups. The high-fidelity gravity perturbation model computations and the state integration are carried out on the GPU. The CPU is responsible for initializing the force models, defining the initial conditions for the current simulation, and managing the uplink and downlink of data to and from the GPU. The algorithm is designed to be general so it can readily apply to other similar classes of problems.

Figure 81 depicts the general algorithmic model. The algorithm starts by initializing the state vector of the objects to be integrated on the CPU. This initial state for each object is passed to the GPU (via a CUDA feature called zero-copy) where the actual integration is performed. The CPU is also responsible for loading and transferring the FIRE and Fetch model coefficients and other force model parameters to the GPU during its initialization phase; more details are presented later. One thread per
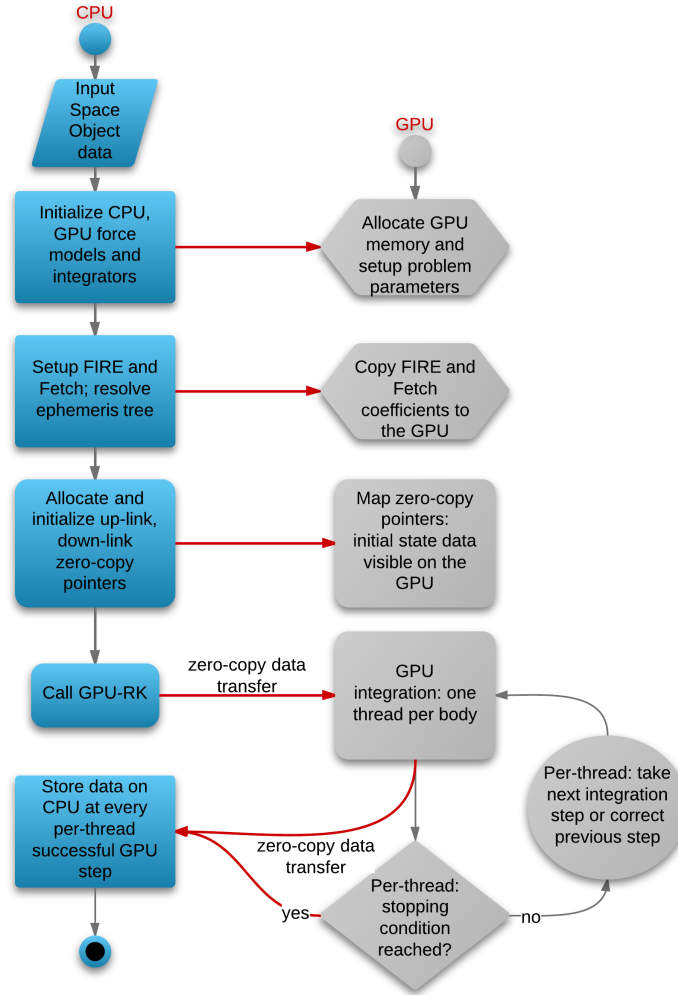
Figure 81: General Algorithmic Model

spacecraft strategy is adopted, hence, the number of threads launched on the GPU is equal to the number of objects to be integrated. Multiple objects are integrated in parallel with each thread integrating each object serially. As the computational model enables each thread to enjoy the speedups from both the FIRE and FETCH models, the final theoretical speedup from the overall simulation is multiplicative in nature and is given by $S = S_p * S_a$, where S represents the total speedup, S_p represents the speedup due to parallelization on the GPU and S_a represents the acceleration speedups due to FIRE and Fetch implemented on the GPU.

At each integration step the new position vector for each of the objects is transferred on the fly to the CPU via zero-copy. Apart from initializing the state vector, the CPU is also responsible for determining the maximum number of steps possible on the GPU based integrator, which is determined by the amount of RAM available on the system. Note, that in our current approach the number of objects that can be integrated is not limited by the GPU memory, hence allowing large scale simulations. The GPU integrator is the Runge-Kutta 54 variable step integrator with coefficients adopted from Ref.[41] More details on the GPU based integrator are given in the next section.

6.5   GPU based Runge-Kutta integrator

To exploit the massive parallelism across multiple spacecraft a GPU based variable step integrator (GPU-RK) is implemented. For simplicity, the current work adopts a Runge-Kutta 54 integrator on the GPU. The integrator is similar to its CPU counter part[41] and is capable of doing computations in full IEEE compliant double precision arithmetic. The input force model maybe general in nature and can be implemented in CUDA or the C programming languages.

GPU-RK has been designed for maximum double precision performance by taking advantage of the latest hardware and software innovations on the GPU. The integrator takes advantage of both the L1 and L2 cache memories present on the FERMI GPU architecture, thereby increasing the performance by up to 20% in favorable cases. Furthermore, as each thread is responsible for integrating its own trajectory, this leads to an embarrassingly parallel implementation, which minimizes the inter-thread communication. GPU-RK also utilizes the extra shared memory available on a modern GPU to store the force model parameters and intermediate accelerations.

Storing the state vector after every successful integration step for each thread can quickly exhaust the limited on board GPU memory. A simple way to over come this

problem is by limiting the number of integration steps on the GPU and calling the GPU integrator multiple times until all objects are integrated for the requested time of flight. Such an approach would require an estimate of the number of integration steps for a given flight time and would change depending on the initial state vector of the object being integrated. This not only increases the implementation complexity but also makes the integration process inefficient.

Another way to overcome this problem is by using the zero-copy feature of CUDA. Zero-copy allows mapping of a host (CPU) memory pointer directly on the GPU. Though slow, this allows each GPU thread to directly read or write data from or to the host memory. On the host side this pointer is accessed like any other variable on the CPU, with any change being immediately visible to every thread on the GPU. Apart from freeing the limited GPU memory, zero-copy also simplifies the memory management process, eliminating the need to explicitly transfer memory between the CPU and GPU. The amount of data that can be allocated at a given time is limited by the amount of RAM present in the system. As zero-copy is limited by CPU-GPU PCIe bandwidth it is only preferred if each thread reads and writes once to the mapped memory pointer. The peak theoretical PCIe bandwidth for transfers between host and device in each direction is ∼6 GB/sec, while the peak GPU memory bandwidth for the Tesla M2090 GPU (with ECC off) is 177 GB/sec. More on zero-copy can be found in the NVIDIA programming guide available here. [1]

The methodology proposed here uses zero-copy for 1) mapping the initial per-object state vector data to the GPU and 2) for writing per-object state vector data at each successful integration step. Given the high compute-to-communication (device to host) ratio of the force model, the impact due to zero-copy on the final performance is small. Using zero-copy frees the GPU memory, enabling integration of large numbers of objects on the GPU, only to be limited by the amount of RAM available on the

---

[1]http://docs.nvidia.com/cuda/index.html

144

system.

6.6   High-Fidelity Gravity Perturbation Model

For high-fidelity spacecraft integration it is essential to take into account drag, solar radiation pressure, the lunisolar perturbation effects and the high order gravity field of the Earth. Hence, the proposed tool takes advantage of a high-fidelity gravity perturbation model given by Eq. 97 during the integration process.

$$\vec{a}_{tot} = \vec{a}_{eph} + \vec{a}_{grav} + \vec{a}_{drag} + \vec{a}_{srp} \tag{97}$$

where $\vec{a}_{grav}$ is given by Eq. 98

$$\vec{a}_{grav} = \vec{a}_{(twobody+J_2)} + \vec{a}_{fetch} \tag{98}$$

$\vec{a}_{fetch}$ refers to the high order gravity acceleration calculated via the Fetch gravity model (see chapter 3), $\vec{a}_{eph}$ is the lunisolar perturbations calculated via FIRE (see chapter 4), and $\vec{a}_{drag}$ and $\vec{a}_{srp}$ represents the atmospheric drag and the solar radiation pressure (SRP) based accelerations calculated via a simple cannon-ball model.

In this study only static gravity terms are considered. Low order solid and ocean tides could be simply added via time dependent low degree and order spherical harmonic representations. Other relevant high fidelity forces that are not considered in this study include 1) high degree and order solid and ocean tides 2) general relativity perturbations, and 3) attitude dependent SRP and drag models. The inclusion of such terms is considered beyond the current scope and will be left to future work.

Both the FIRE and Fetch models are computed on the GPU and details of their implementation are given in the next section. The computations are performed in a non-rotating Earth centered frame and the required Earth orientations are also obtained through the FIRE model.

6.6.1 Lunisolar Ephemeris and Earth Orientation

Precise lunisolar ephemeris positions are essential for computing perturbation forces due to the Sun and the Moon. The Earth's orientation is necessary for use with sectoral and tesseral gravity terms when a non-rotating integration frame is utilized. If a body fixed integration frame is utilized, the Earth orientation is necessary to transform the lunisolar positions. A state-of-the art and widely accepted source for both lunisolar and body orientation information is the JPL SPICE system.[2] The FIRE ephemeris system, developed in chapter 4, is an attractive alternative that uses SPICE as its source data, and is custom built for problems that favor higher speed and smooth derivatives.

FIRE is capable of providing orders of magnitude in speed improvement (see chapter 4) over the SPICE model.[2] It also provides continuous and analytic first and second derivatives of states and orientation matrices. The higher order derivatives as well as offering continuity are attractive features for large scale high-fidelity trajectory optimization and orbit estimation applications. Given these features, FIRE is adopted as the GPU ephemeris model and is used to compute lunisolar positions and Earth orientations. A direct implementation of the FIRE tree (see Fig. 60, chapter 4) traversal algorithm is not favorable for the GPU architecture. Excessive thread branching present in a general tree algorithm leads to warp serialization (see NVIDIA programming guide [1]) and results in slow GPU performance. Instead, for the current work, the FIRE tree is resolved on the CPU and the coefficients are transformed relative to the center of integration before being copied to the GPU global memory. Subsequently, only one double to integer conversion combined with Eq. 86 (see chapter 4 for details) is needed to compute the lunisolar ephemeris and Earth orientation.

---

[1]http://docs.nvidia.com/cuda/index.html

6.6.2 High Order Geopotential

The conventional approach for computation uses spherical harmonics (SH) geopotential, which is known to be computationally slow when evaluated at high orders.[27, 101] It is simply not practical when accounting for a large number of objects to use high degree and order SH, as the current best practices are usually limited to 36×36.[43] The Fetch model, proposed in chapter 3, efficiently trades memory for speed and delivers orders of magnitude in speedup over state of the art SH approaches. The Fetch model achieves many desirable properties like: 1) it is non-singular, continuous and smooth 2) it is adaptive in terms of local vs. global resolution 3) it has a residual error profile that is dynamic and conservatively in the noise of the accuracy of the SH fitting function.

As part of this study the Fetch gravity model is implemented on the GPU. The base SH harmonics field corresponds to the GRACE GGM03C gravity model. The underlying FETCH algorithm is redesigned to take advantage of the L1 and L2 caches available on the FERMI GPU architecture by keeping in mind the cache locality of the coefficient lookup table. As the algorithm involves looking up polynomial coefficients, it performs significantly faster if threads in groups of 32 are working on objects that are spatially close to each other. Instruction level parallelism is encouraged to some extent by the reordering of independent instruction so as to reduce read-after-write register dependencies when computing high order gravity perturbations.

6.7 Tool Configuration

Table 31 lists the current tool configuration. The algorithm is sensitive to the number of threads per block parameter, which defines the number of threads on the GPU capable of accessing the same shared memory. As the number of threads per block increases, so does the amount of required shared memory. Furthermore, the GPU code uses almost the maximum number of registers (63 out of 64 for FERMI, and

121 out of 128 for pre-FERMI architecture) possible per thread block, and achieves a theoretical occupancy of 33%. Even though the force model affords large amounts of computations per thread, the number of global memory transactions still have a dominant effect on the runtime performance. As the Fetch and FIRE coefficients are accessed from the global memory, their access pattern has a significant impact on this tool's runtime performance. Coalesced and cache friendly access patterns can be expected to be up to 4-6 times faster than totally random access patterns. Also, note that due to the memory transactions required for the coefficients lookup, the Fetch model performs slower on the GPU than the CPU.

The CPU and GPU variable step integrators are both fixed to a normalized tolerance of 1E-11. The tool interfaces with the user via a Fortran name-list input file. The user has the freedom to provide the initial state vector for all the objects that are under consideration, or the tool can randomly distribute the objects over a range of space. The gravity model can also be changed to a lower or higher order Fetch model, if needed.

Table 31: Current Tool Configuration

Property name	Type
Ephemeris model	Sun + Moon perturbation model (via FIRE)
Gravity field resolution	70×70, 156×156, 360×360 (deg, via Fetch)
Drag, SRP model	Point mass
Threads per block	64

Table 32 gives an overview of the current test hardware.

6.8 Performance Evaluation

In this section the performance of the tool is compared to a state-of-the-art serial CPU implementation using the SH gravity field model[101] and JPL's SPICE[2] ephemeris. Table 33 lists the various aspects considered for defining the speedup parameter.

With these points in mind, speedup is simply the ratio of GPU execution time over

Table 32: Test hardware specifications

Component type	Component
CPU	Intel Xeon X5650 @ 2.67 Ghz
Operating system	Linux X86_64
GPU	1 TESLA M2090
# Cuda cores	512
CPU memory	48.0 GB
GPU memory	6 GB
CPU compiler	Intel Fortran 12.0
GPU compiler	NVCC 5.0

Table 33: Tool configuration

Property	CPU implementation	CPU-GPU implementation
Higher order gravity field	SH	Fetch
Lunisolar perturbations	SPICE, FIRE	FIRE
Integrator	RKF-54, DOPRI-78	GPU RKF-54
Runtime	scaled	fully computed

the scaled CPU execution time. Simulations on the CPU take on the order of days to years to complete (on a single workstation CPU), hence, for the current study the CPU propagation time is calculated for a small representative set of objects (1024) for flight times equal to their orbital periods and then scaled to match the number of objects and flight times simulated on the GPU. To evaluate the performance of the tool two cases are considered:

- Case 1: Object initial conditions are densely packed to simulate a distribution about a reference orbit (LEO).

- Case 2: Object initial conditions are randomly distributed over the LEO/MEO environment.

6.8.1   Case 1: Dense Distribution

In this case, objects are initially clustered together in a random fashion forming a uniform 6D distribution, akin to an uncertainty distribution about a single object.

Table 34 gives the nominal 6 states (position, velocity) and gives the ranges on each state. Table 35 gives the average orbital elements of the densely packed distribution.

Table 34: Three state nominal value and range (body-fixed frame)

State	Nominal Value	Range
x	5932.27 (km)	$\pm$ 0.5
y	1445.81 (km)	$\pm$ 0.5
z	3104.89 (km)	$\pm$ 0.5
u	-4.35 $(\frac{km}{sec})$	$\pm$ 0.002
v	3.18 $(\frac{km}{sec})$	$\pm$ 0.002
w	6.83 $(\frac{km}{sec})$	$\pm$ 0.002

Table 35: Initial condition (body-fixed frame) for trajectory integration

Orbital Parameter	Value
Periapsis (r_p)	6950.00 (km)
Eccentricity (e)	0.30
Inclination (i)	65 (degrees)
Longitude of ascending node (Ω)	0 (degrees)
Argument of periapsis (ω)	9 (degrees)
True anomaly at epoch (ν)	0 (degrees)

For the performance tests, the flight time varies from 72 minutes to 1.5 days. As the objects are densely packed, they all take approximately the same number of integration steps. The number of objects on the GPU varies from 1,024 to 131,072. Three different SH fields of degree and order of 70, 156, and 360 are considered.

Figures 82 to 84 show the case 1 speedup contours for various SH field sizes when using the RKF-54 integrator and SPICE on the CPU. As the objects are densely packed, the GPU L1 and L2 cache memory misses are minimized making force evaluations the most efficient. Average speedups of 1,629.29, 8,801.75, and 67,373.36 are achieved, for the three SH field sizes, respectively, when simulating 16,384 objects. Further increasing the number of objects, leads to little or no change in these speedup numbers. Figures 85 to 87 show the speedup contours when SPICE is replaced by FIRE on the CPU. Average reduction in speedup when moving from SPICE to FIRE
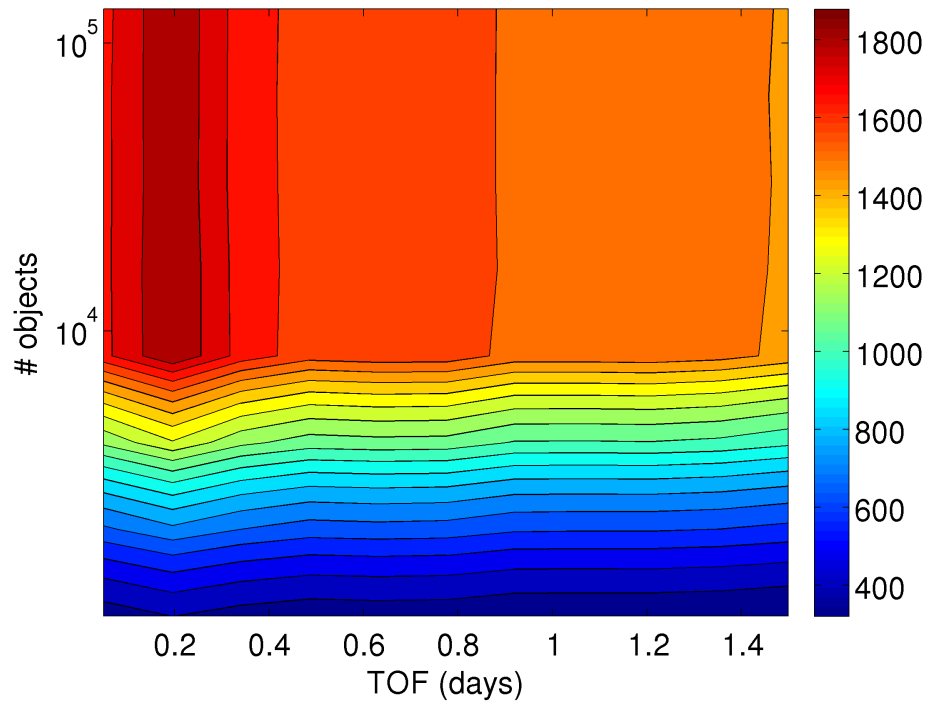
Figure 82: Case 1: Scaled speedup, RKF-54, SPICE, 70×70 SH field
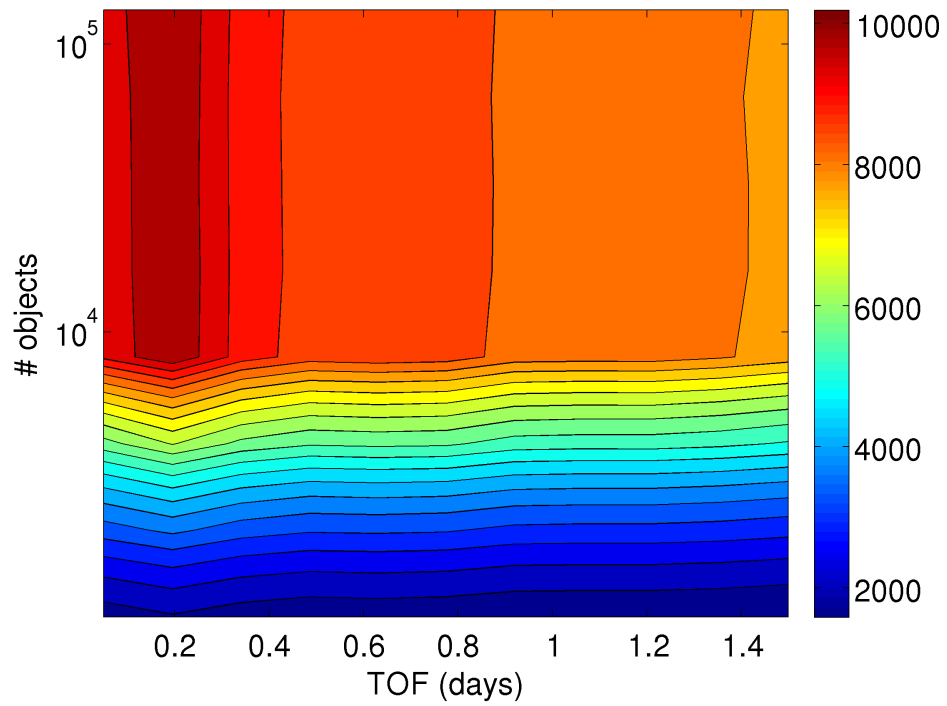

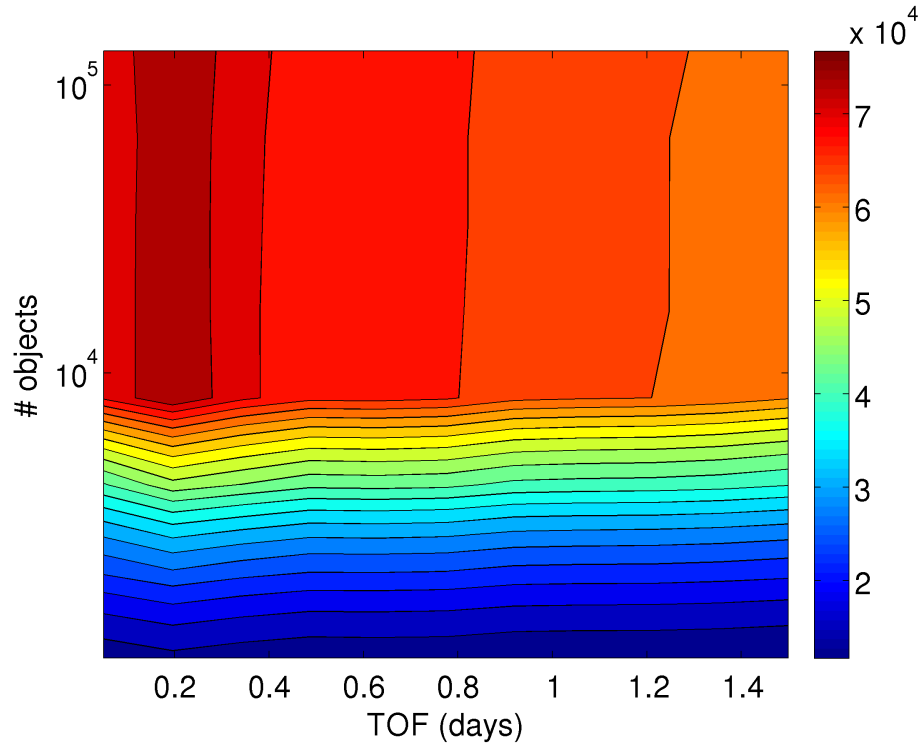**Figure 83:** Case 1: Scaled speedup, RKF-54, SPICE, 156×156 SH field

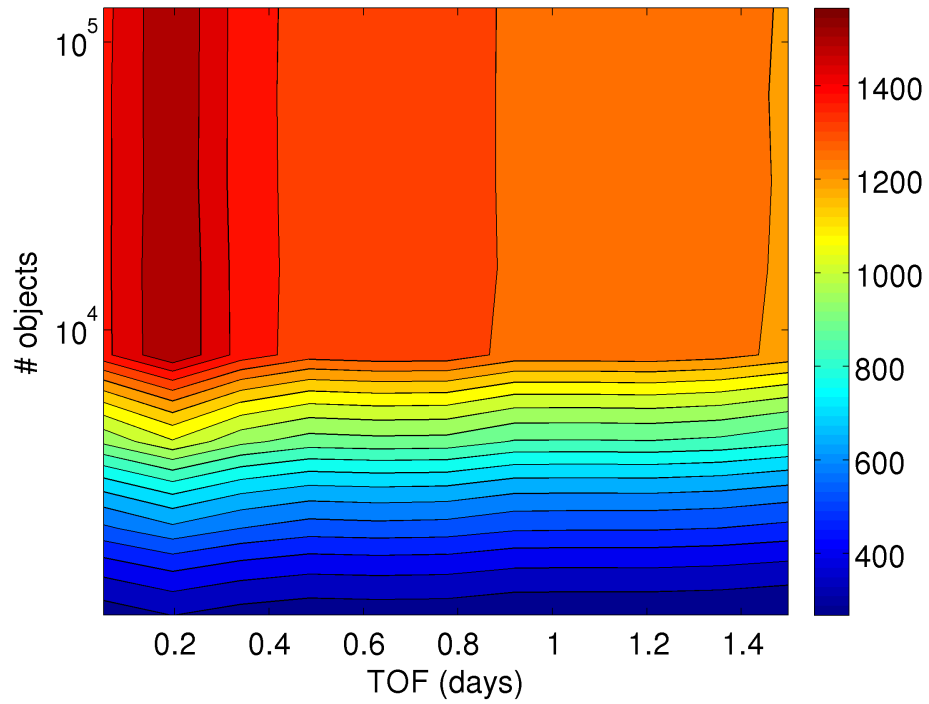**Figure 84:** Case 1: Scaled speedup, RKF-54, SPICE, 360×360 SH field

Figure 85: Case 1: Scaled speedup, RKF-54, FIRE, 70×70 SH field
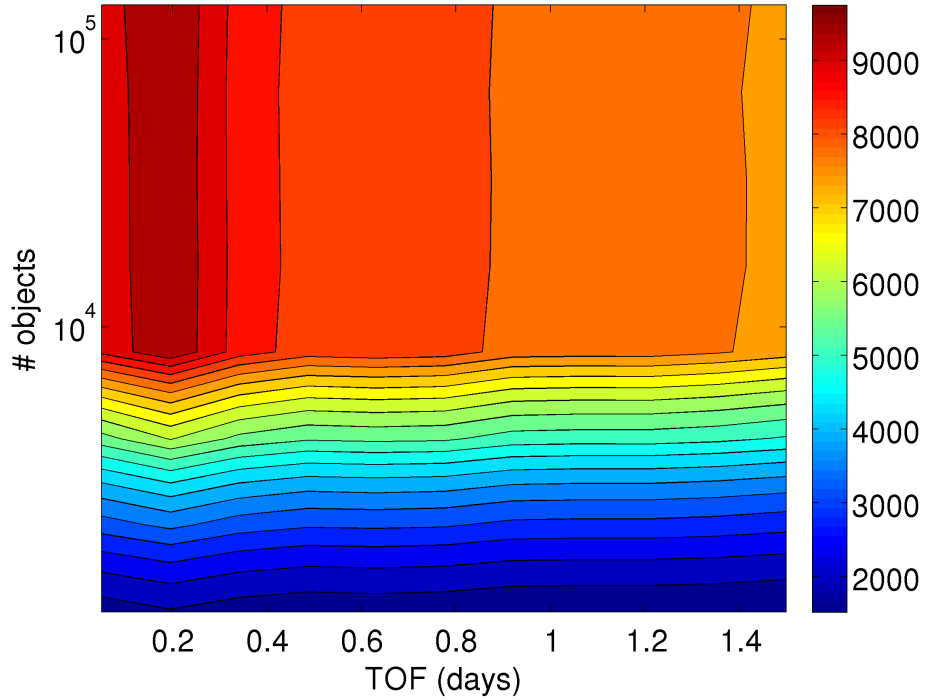
Figure 86: Case 1: Scaled speedup, RKF-54, FIRE, 156×156 SH field

(on the CPU) for SH field sizes of 70, 156 and 360 is found to be 16.6%, 4.30%, and 0.48%, respectively. Increasing the SH field size decreases the gains from using a fast ephemeris computation system like FIRE.

Switching the CPU integration method from RKF-54 to DOPRI-78 reduces the speedup values by a factor of ~ 2 (Figs. 88 to 93). Even though the CPU-GPU comparison is no more "apples to apples" it does motivate one to implement a higher order integrator on the GPU in the future. The higher order integrator would require fewer integration steps, thereby reducing the number of GPU global memory transactions and total storage requirements.

The shape of the speedup contours can be explained by the fact that for a small number of bodies the GPU execution time is bounded by global memory latency. Increasing the number of objects allows the GPU to more successfully hide this memory latency until the code becomes balanced (at ~ 8192 objects). Further increasing the
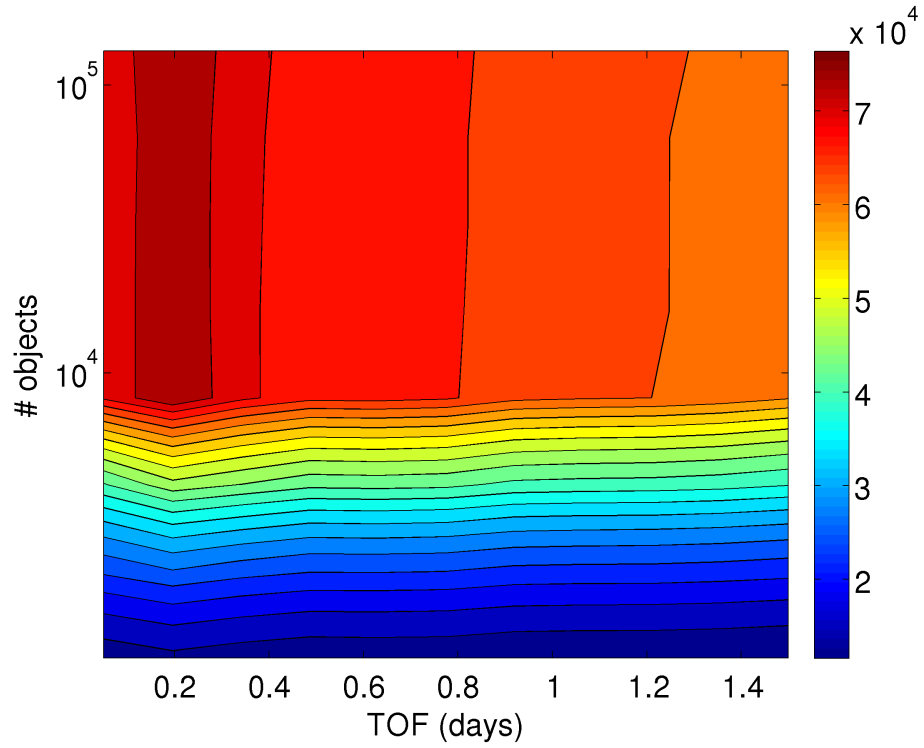
Figure 87: Case 1: Scaled speedup, RKF-54, FIRE, 360×360 SH field
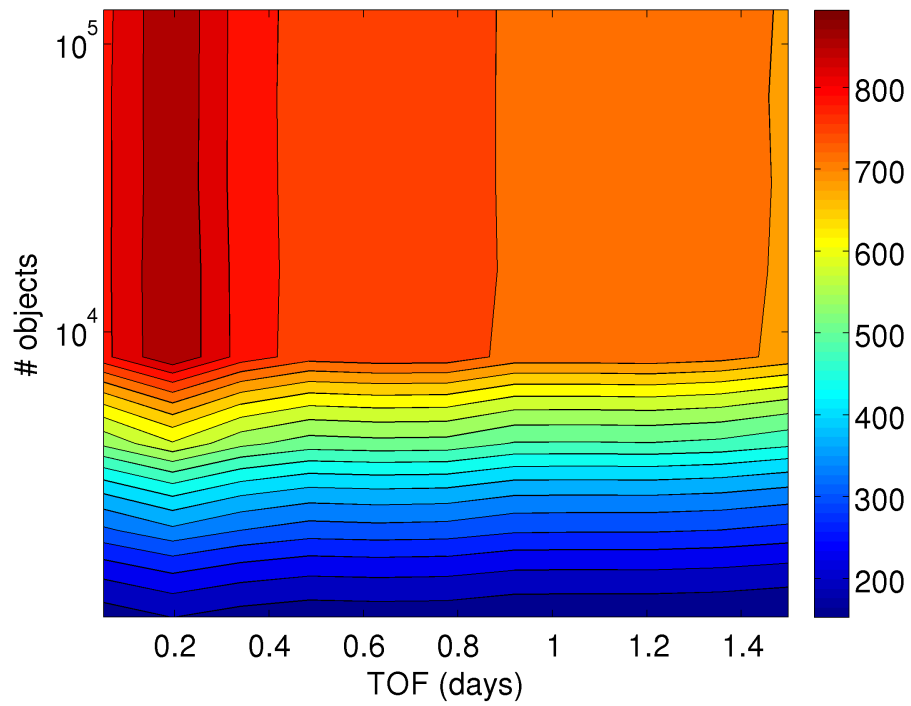
Figure 88: Case 1: Scaled speedup, DOPRI-78, SPICE, 70×70 SH field
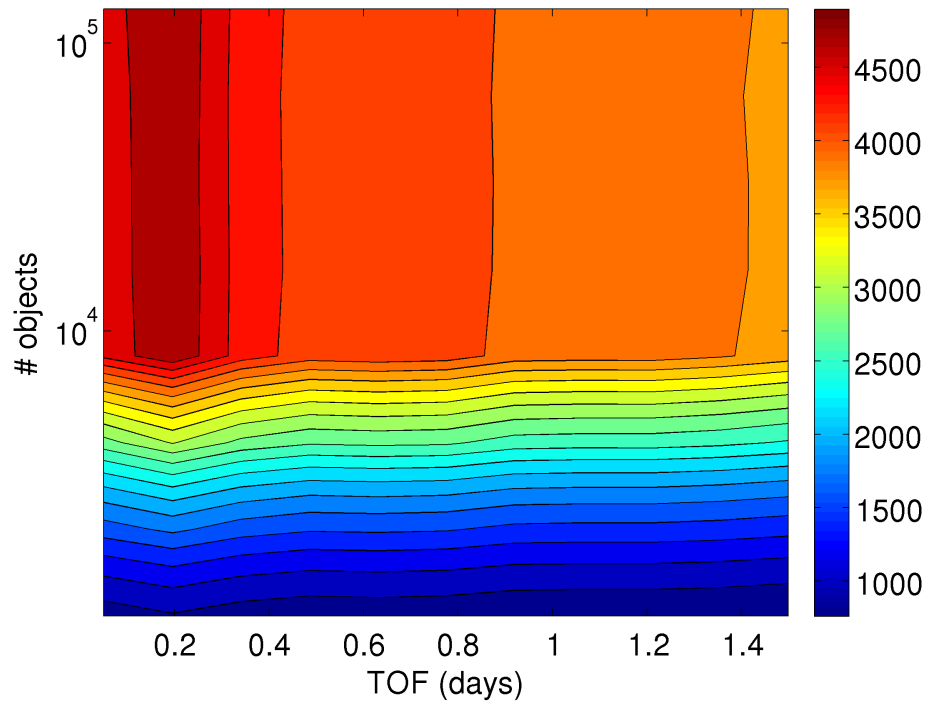
Figure 89: Case 1: Scaled speedup, DOPRI-78, SPICE, 156×156 SH field
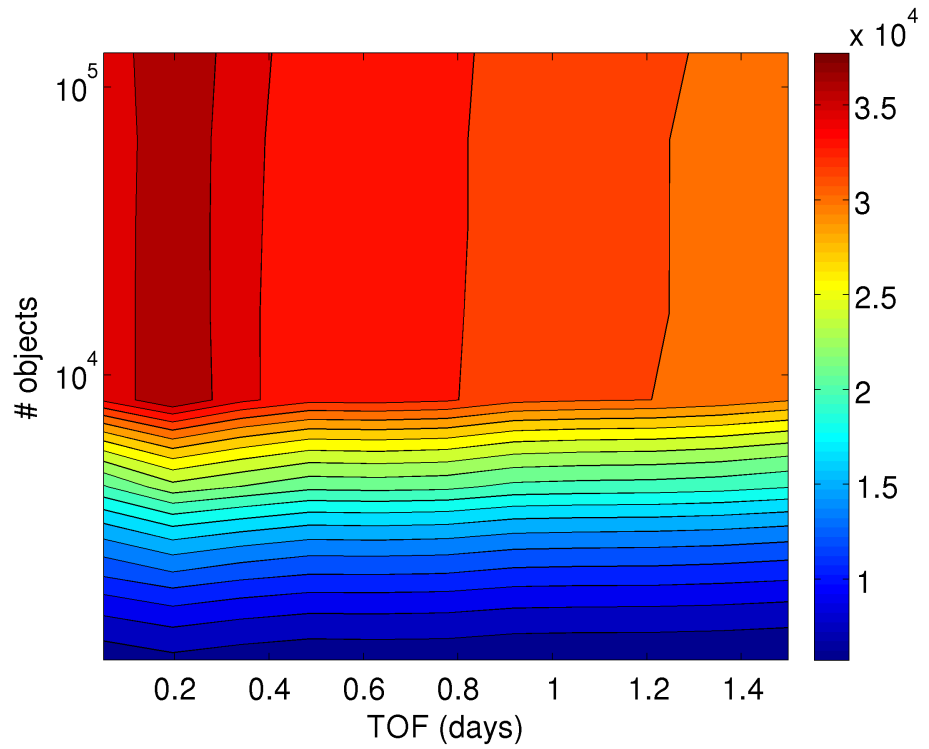
Figure 90: Case 1: Scaled speedup, DOPRI-78, SPICE, 360×360 SH field
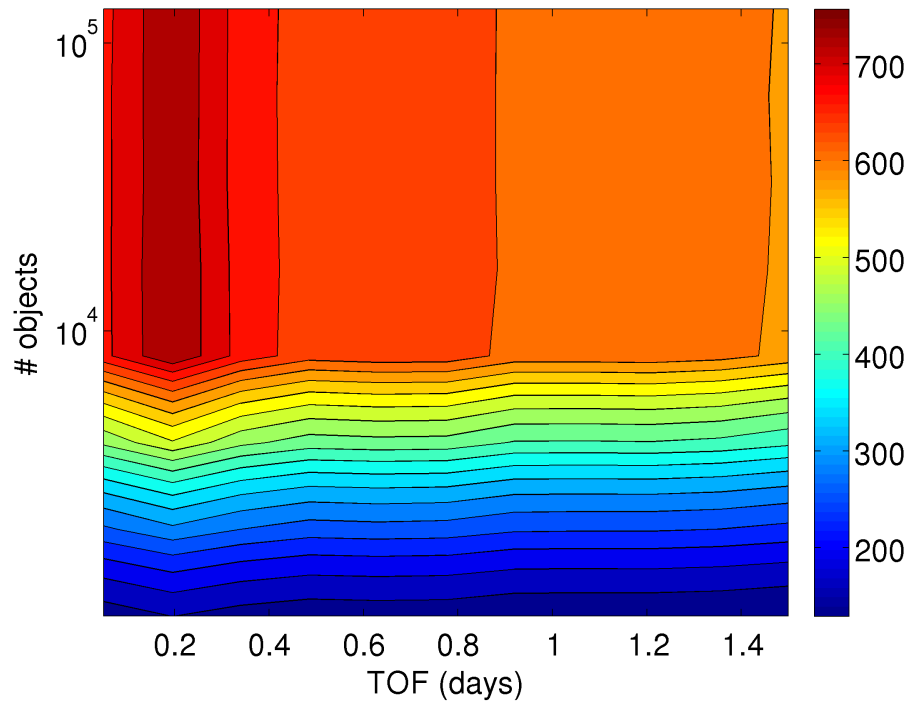
Figure 91: Case 1: Scaled speedup, DOPRI-78, FIRE, 70×70 SH field

Figure 92: Case 1: Scaled speedup, DOPRI-78, FIRE, 156×156 SH field
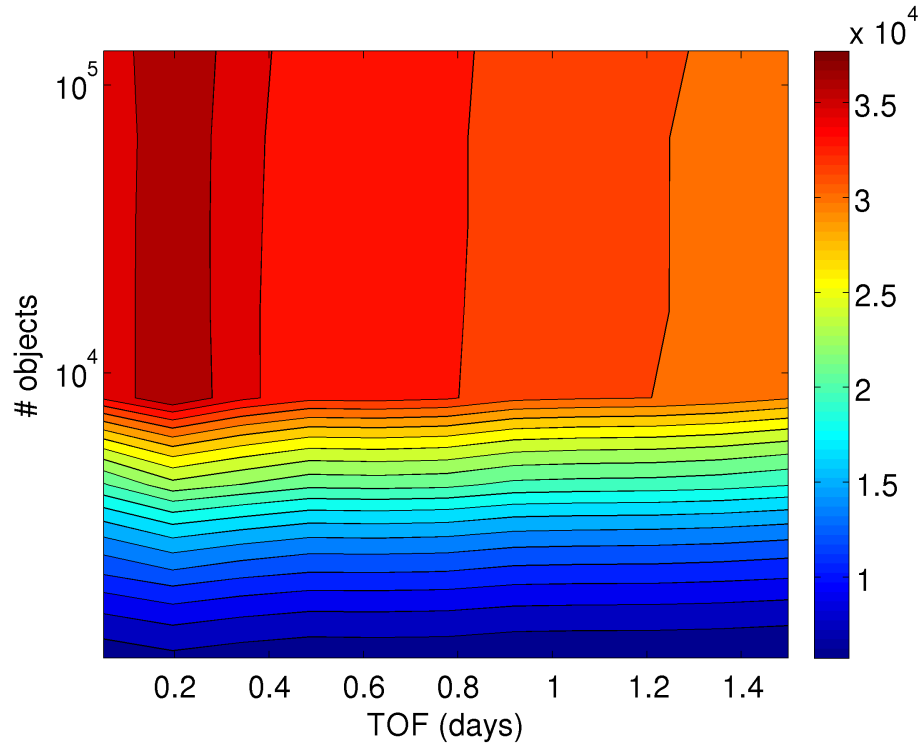
Figure 93: Case 1: Scaled speedup, DOPRI-78, FIRE, 360×360 SH field

number of objects leads to a linear increase in GPU runtime. Also, for large flight times the impact of zero-copy transfers becomes more significant, which leads to a small increase in GPU runtimes.

6.8.2 Case 2: Random Distribution

In this case, the objects are randomly distributed over the LEO environment under the following constraints on their orbital elements:

- Periapsis and apoapsis vary between 6,600 to 10,000 km.

- All angular orbital elements vary between 0 and 360 deg.

Similar to the previous case, the flight time varies from 72 minutes to 1.5 days. The number of objects on the GPU again varies from 1,024 to 131,072.

Figures 94 to 99 show the case 2 speedup contours for various SH field sizes when using the RKF-54 integrator and either SPICE or FIRE, on the CPU. Comparing
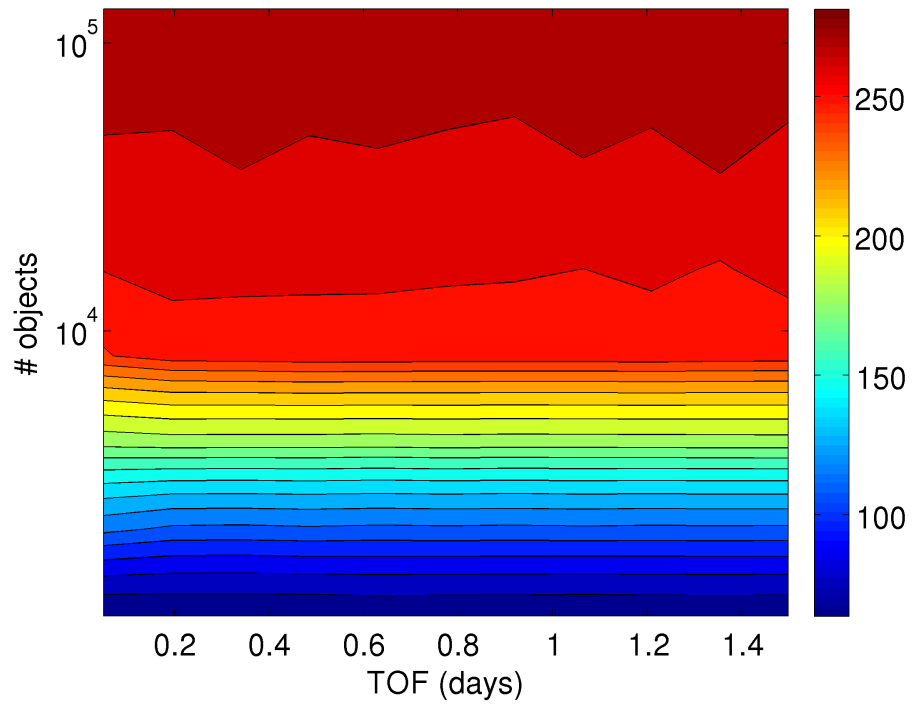
Figure 94: Case 2: Scaled speedup, RKF-54, SPICE, 70×70 SH field
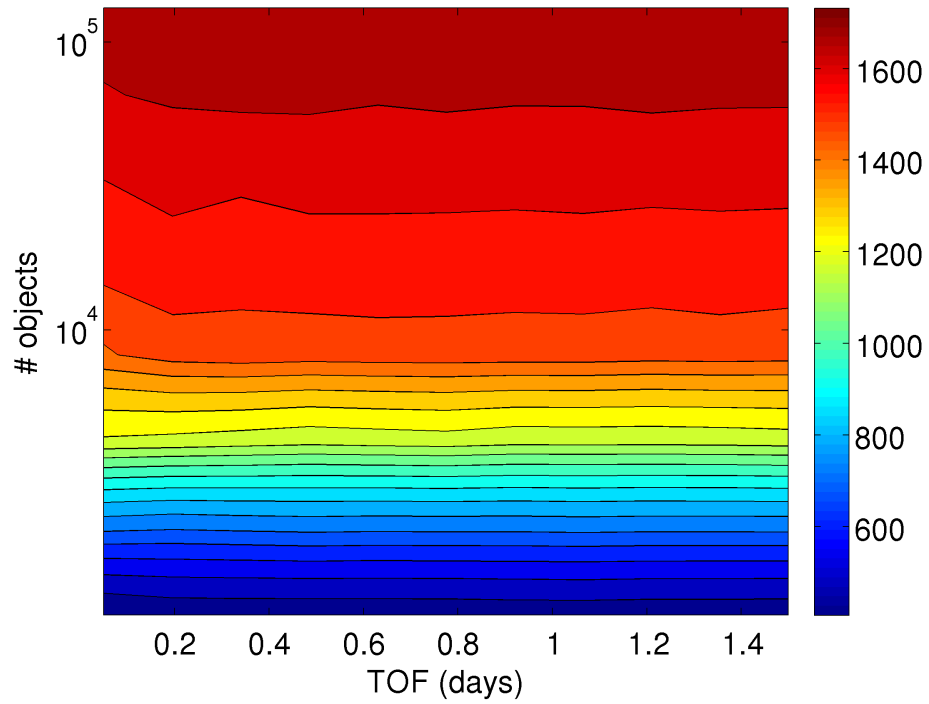
Figure 95: Case 2: Scaled speedup, RKF-54, SPICE, 156×156 SH field
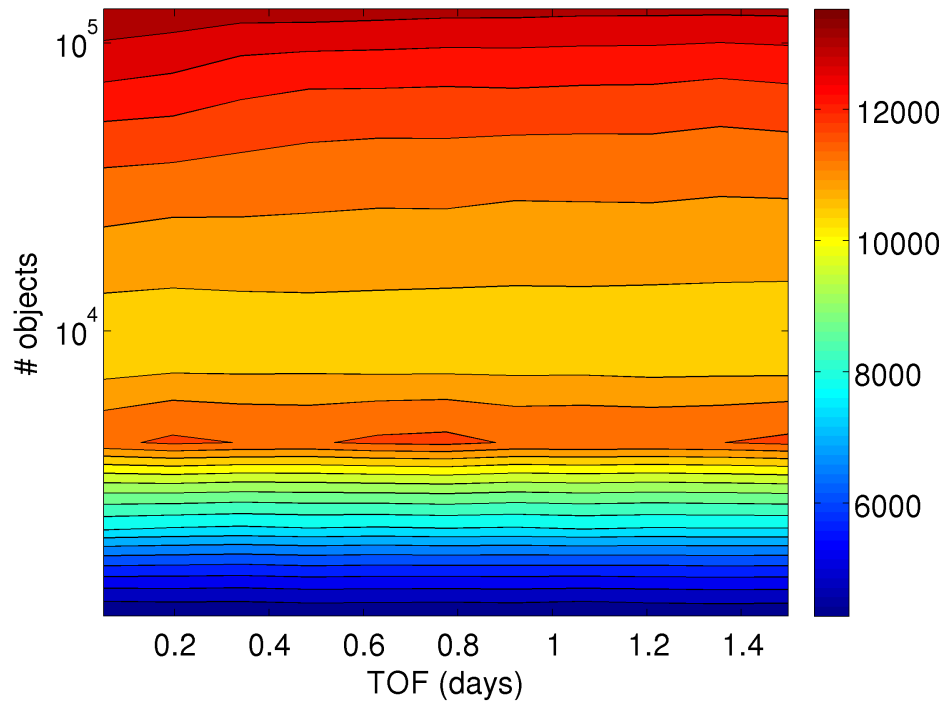
Figure 96: Case 2: Scaled speedup, RKF-54, SPICE, 360×360 SH field
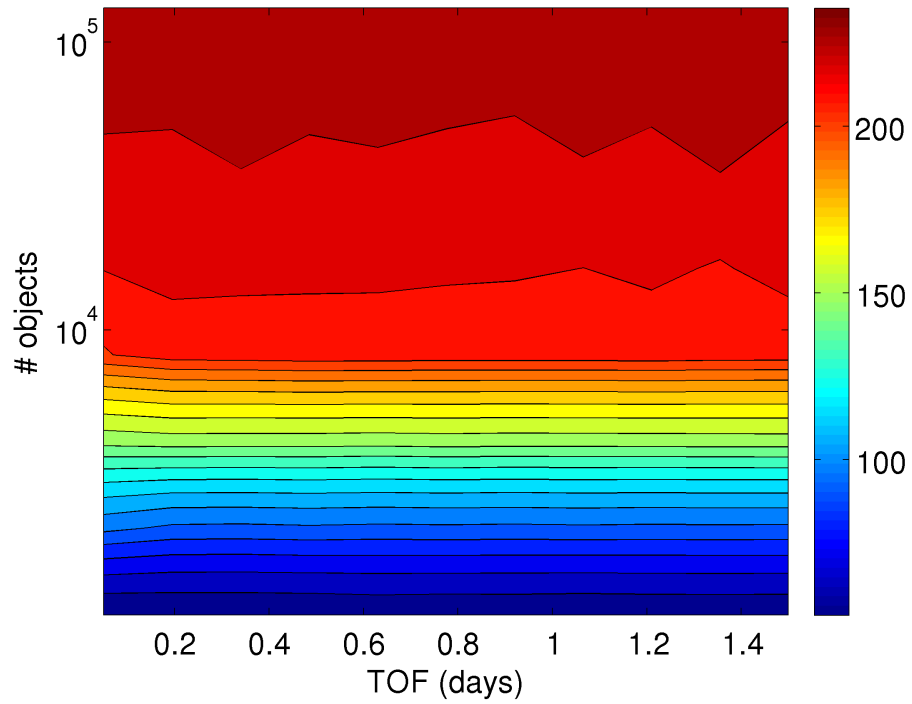
Figure 97: Case 2: Scaled speedup, RKF-54, FIRE, 70×70 SH field

159
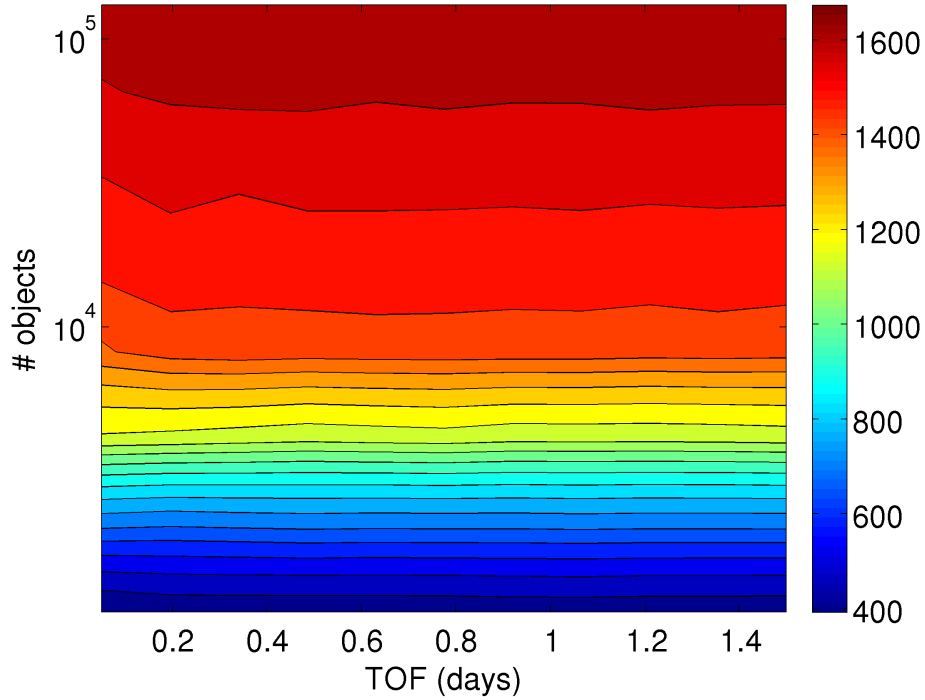
Figure 98: Case 2: Scaled speedup, RKF-54, FIRE, 156×156 SH field

Figs. 84 and 96, it is clear that there is a 3-4 fold decrease in GPU runtime performance which is explained due to two reasons. One, the tool does not take advantage of the fact that the memory is written on the CPU continuously (at each integration step). Hence, the CPU waits until all GPU threads finish integration. Given the varying number of steps across multiple threads, there is a drop in efficiency of the algorithm. Second, the tool does random coefficient look-ups that are required to compute higher order perturbations from the Fetch and FIRE models. Similar to the previous case, switching the CPU integration method from RKF-54 to DOPRI-78 reduces the speedup values by a factor of $\sim$2 to $\sim$3 (Figs. 100 to 105). The contour patterns for this case are simpler compared to the previous case as the GPU runtime is dominated by global memory read and write latency.

The two cases studied provide approximate upper and lower bounds on the performance of the tool, with the best case scenario being the densely packed case and
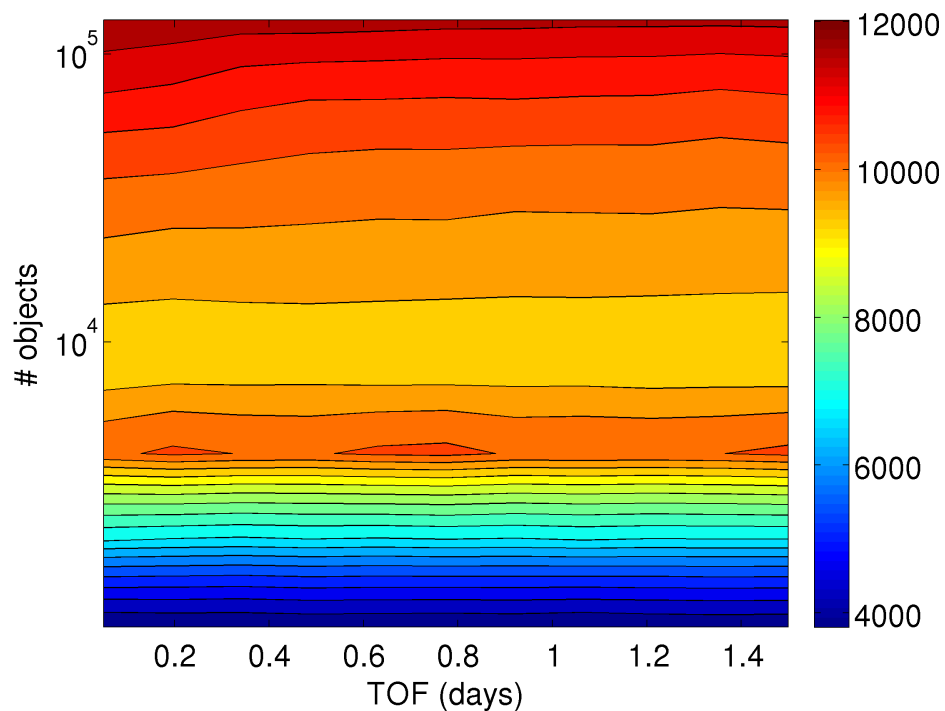
Figure 99: Case 2: Scaled speedup, RKF-54, FIRE, 360×360 SH field
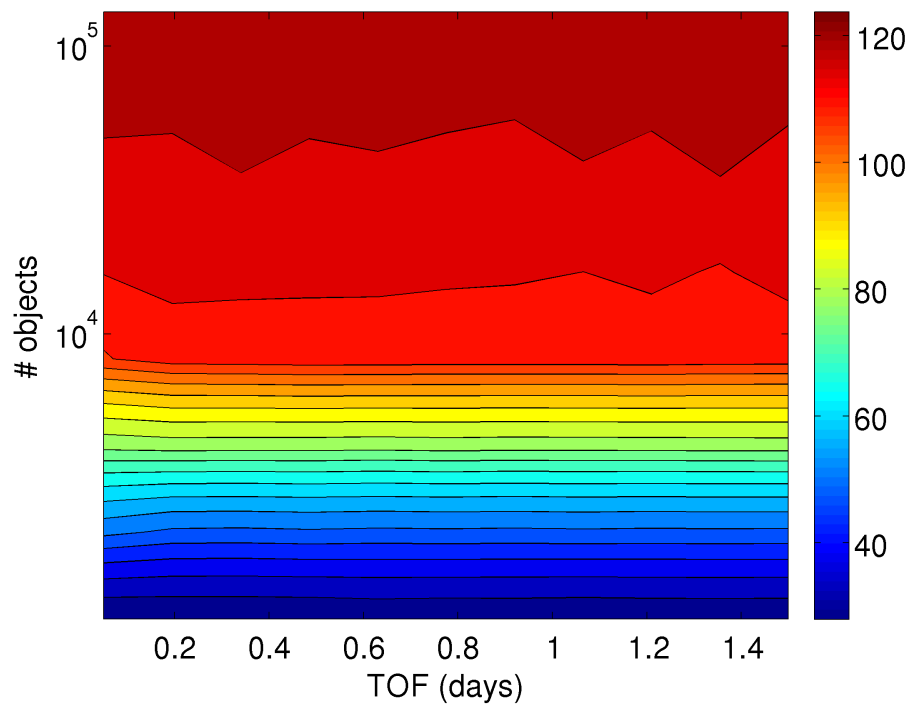
Figure 100: Case 2: Scaled speedup, DOPRI-78, SPICE, 70×70 SH field
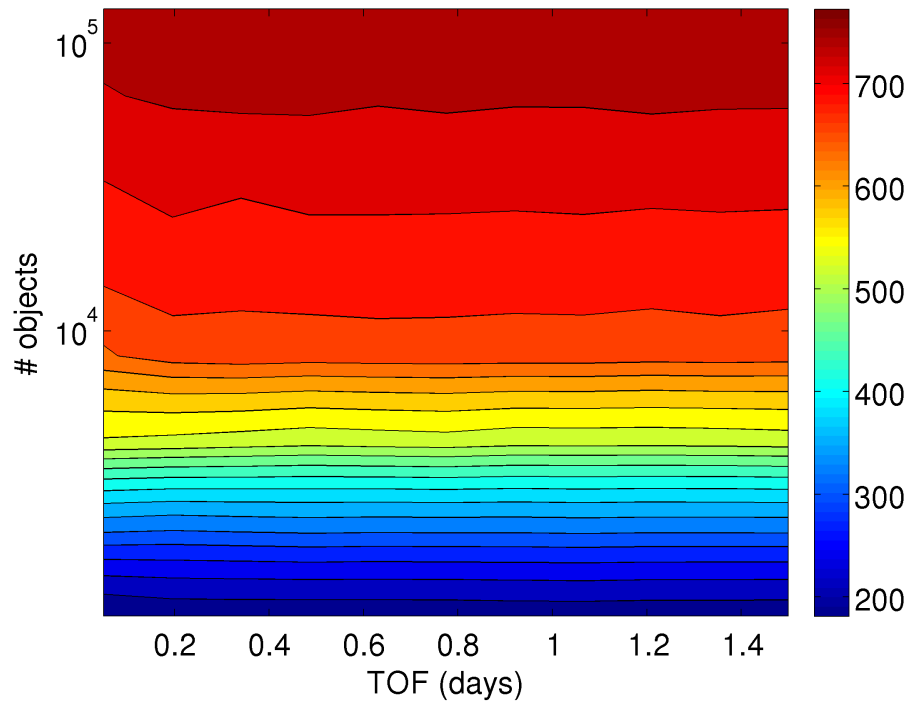
Figure 101: Case 2: Scaled speedup, DOPRI-78, SPICE, 156×156 SH field
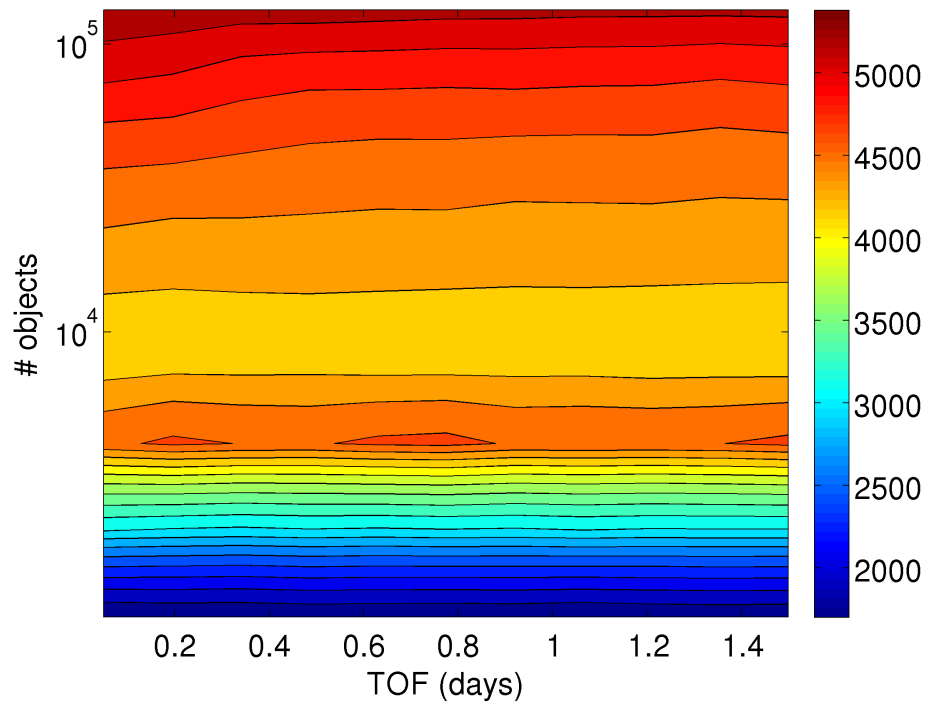
Figure 102: Case 2: Scaled speedup, DOPRI-78, SPICE, 360×360 SH field
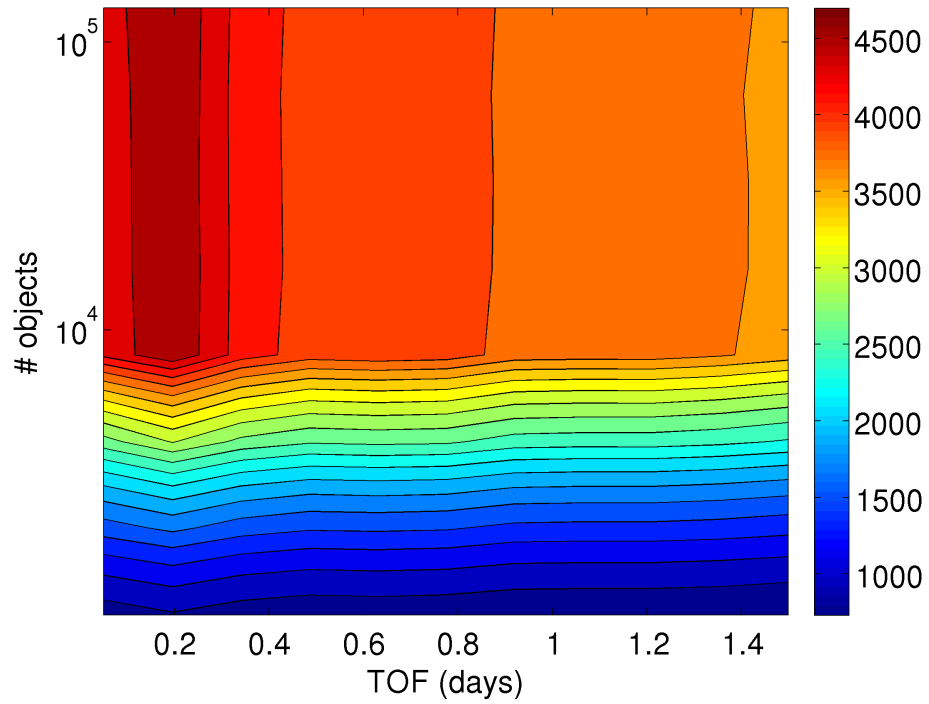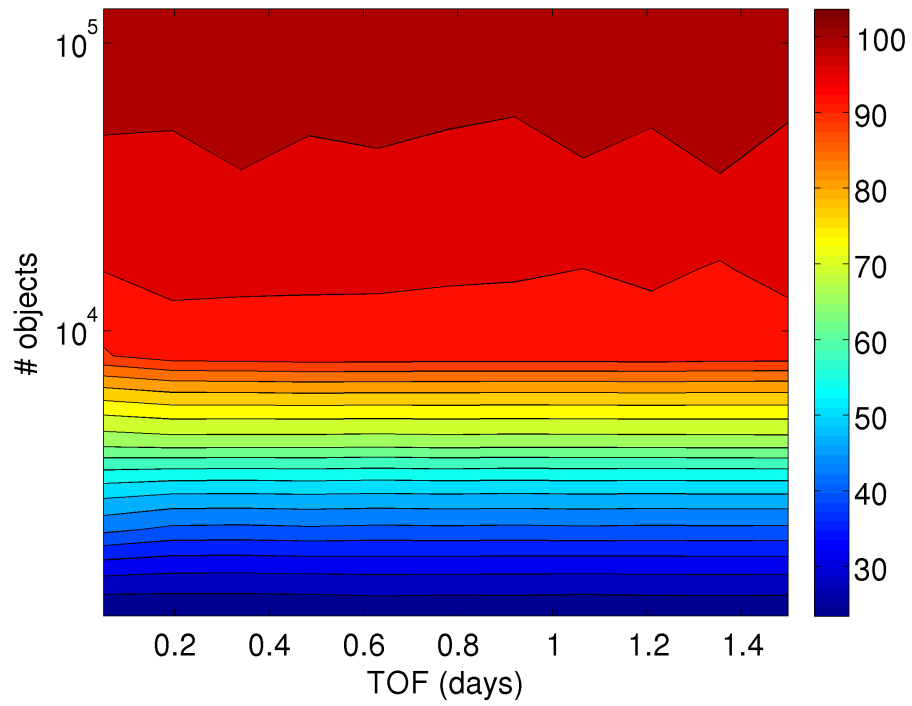
Figure 103: Case 2: Scaled speedup, DOPRI-78, FIRE, 70×70 SH field
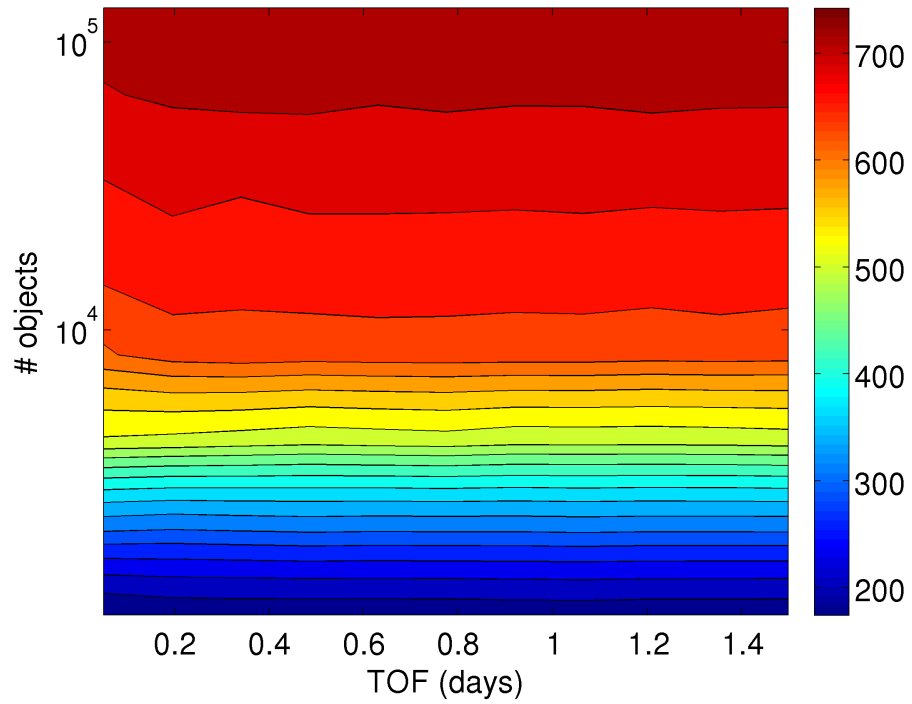
Figure 104: Case 2: Scaled speedup, DOPRI-78, FIRE, 156×156 SH field
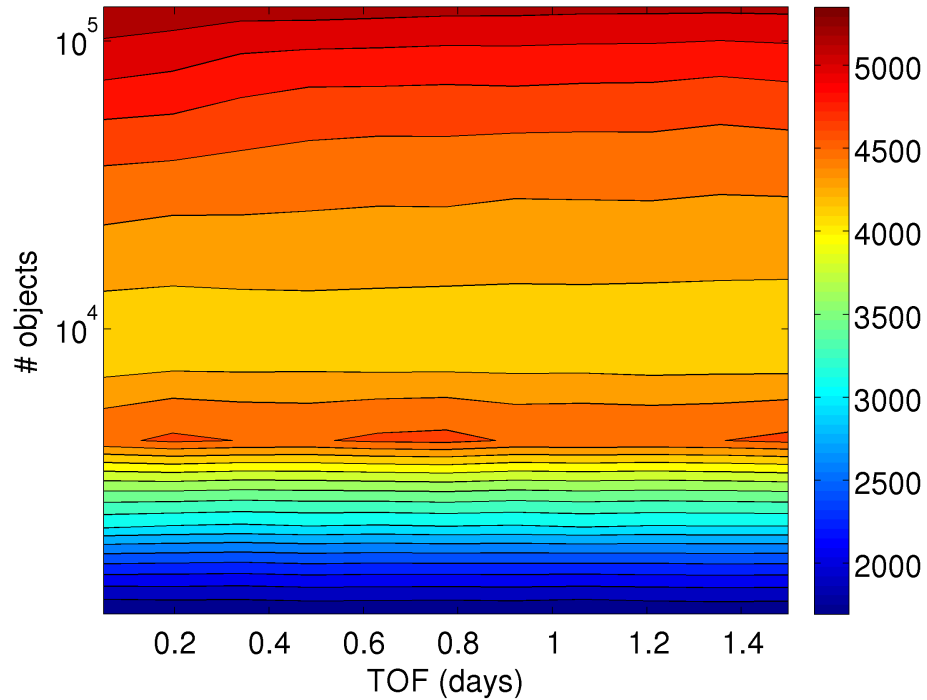
Figure 105: Case 2: Scaled speedup, DOPRI-78, FIRE, 360×360 SH field

the worst case scenario being the randomly distributed case. It should also be noted that typical real world applications like the space catalog consist of bodies which can be grouped (to some extent) depending on their proximity to each other, pushing the speedup values closer to those of case 1.

The speedup values presented in this chapter also reflect the fact the CPU code under utilizes the CPU. Theoretically, if an algorithm utilizes all possible CPU cores (6 for the current setup) to their full potential then the maximum theoretical speedup by using the GPU will be approximately 7 times. Generally speaking, achieving close to peak CPU performance is not possible unless the algorithm maps efficiently to the CPU architecture, takes advantage of the multi-core parallelism and has undergone extensive optimization.
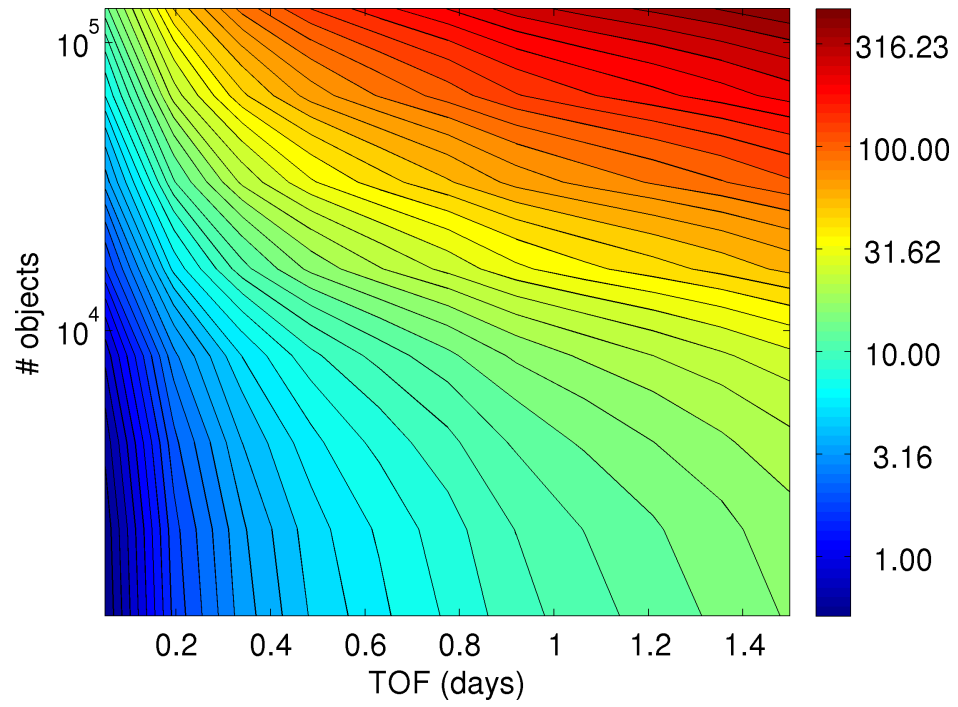
Figure 106: Case 1: absolute runtime (sec), 70×70 SH field
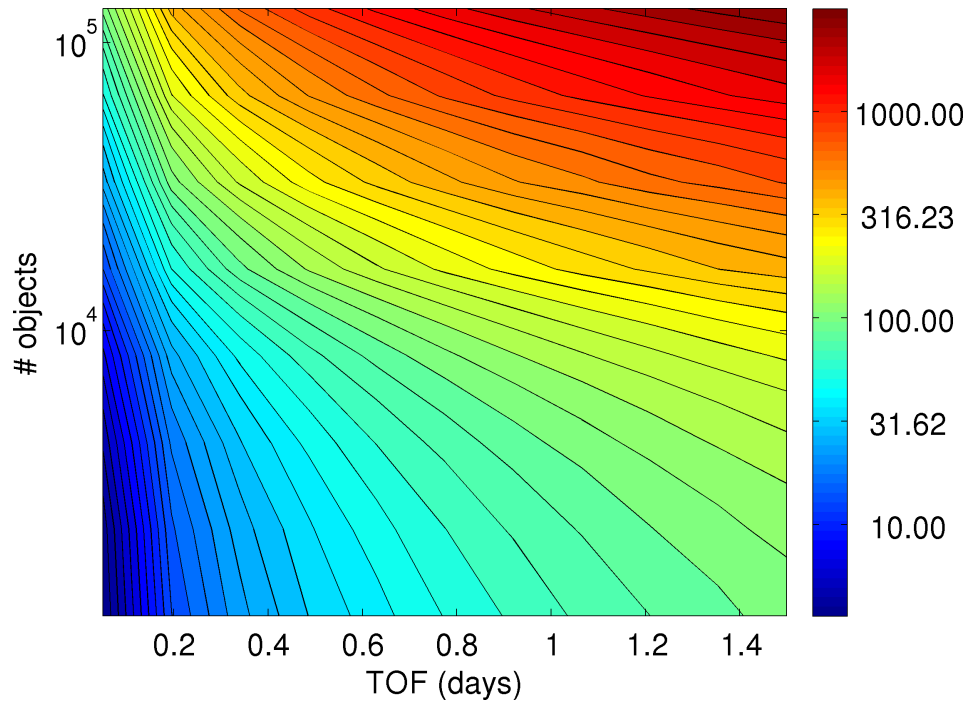
Figure 107: Case 2: absolute runtime (sec), 70×70 SH field
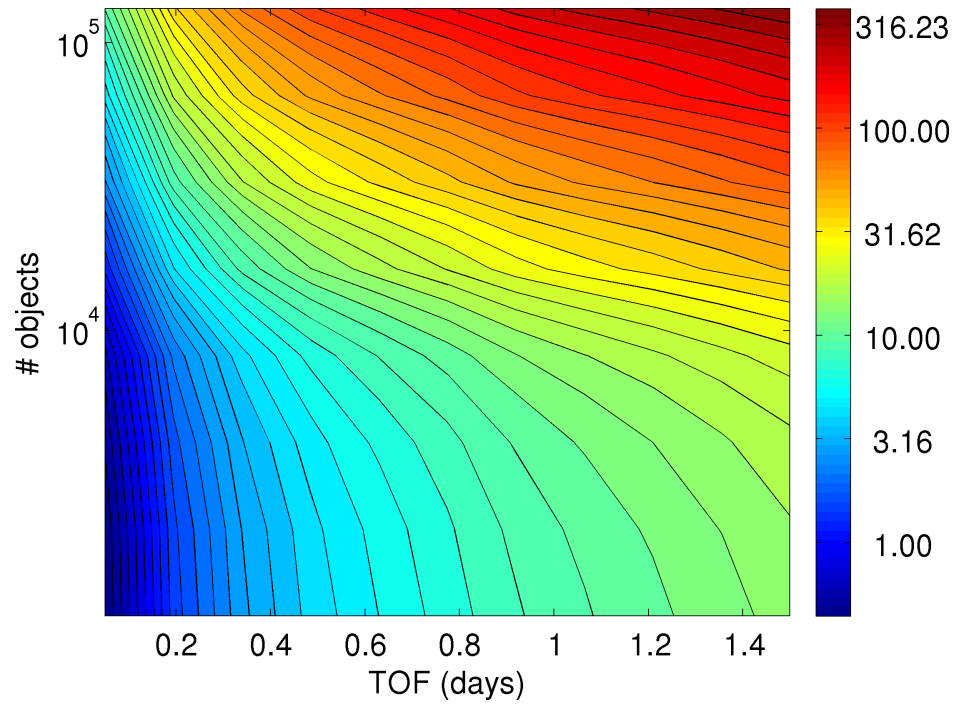
Figure 108: Case 1: absolute runtime (sec), 156×156 SH field
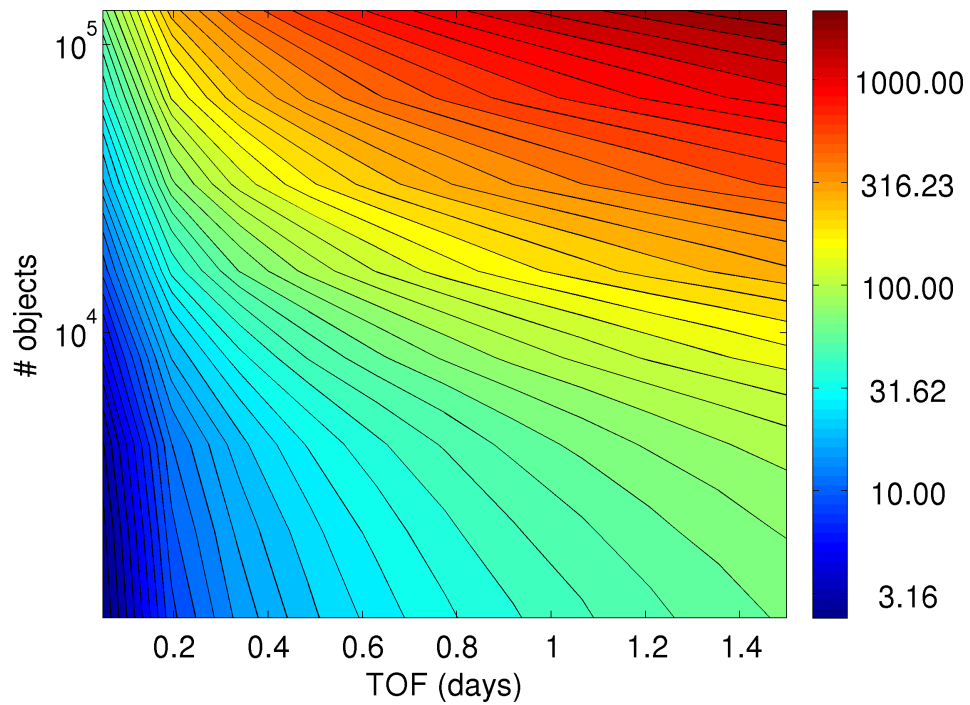
Figure 109: Case 2: absolute runtime (sec), 156×156 SH field

6.8.3 Absolute performance

Figures 106 and 111 show the absolute runtime contours corresponding to cases 1 and 2 and for the three SH field sizes, respectively. Up to 32 GB of RAM is used to
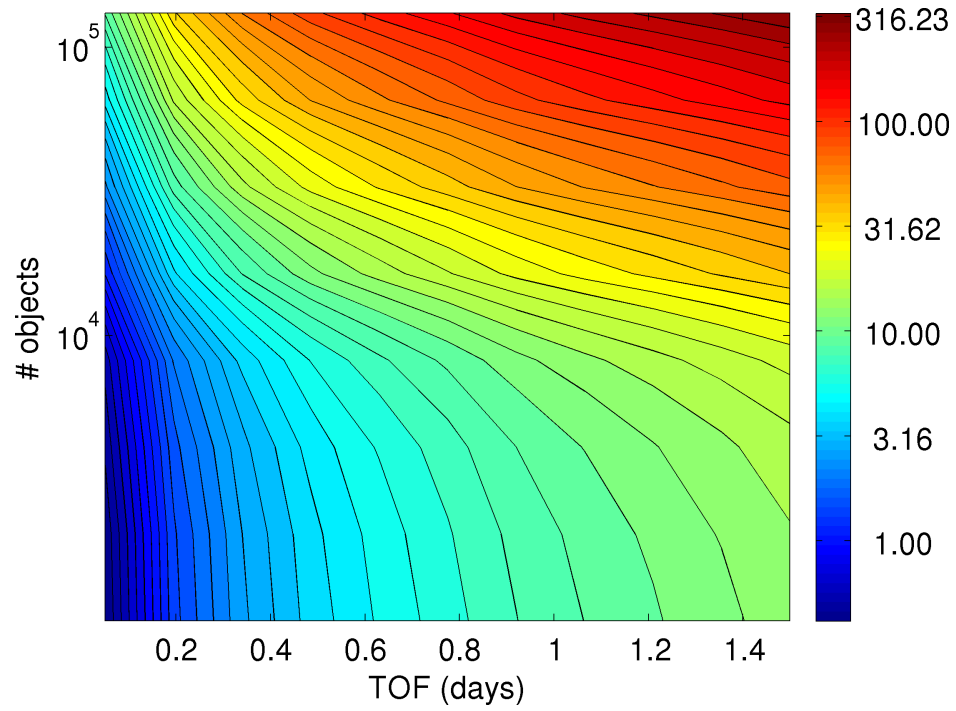
Figure 110: Case 1: absolute runtime (sec), 360×360 SH field
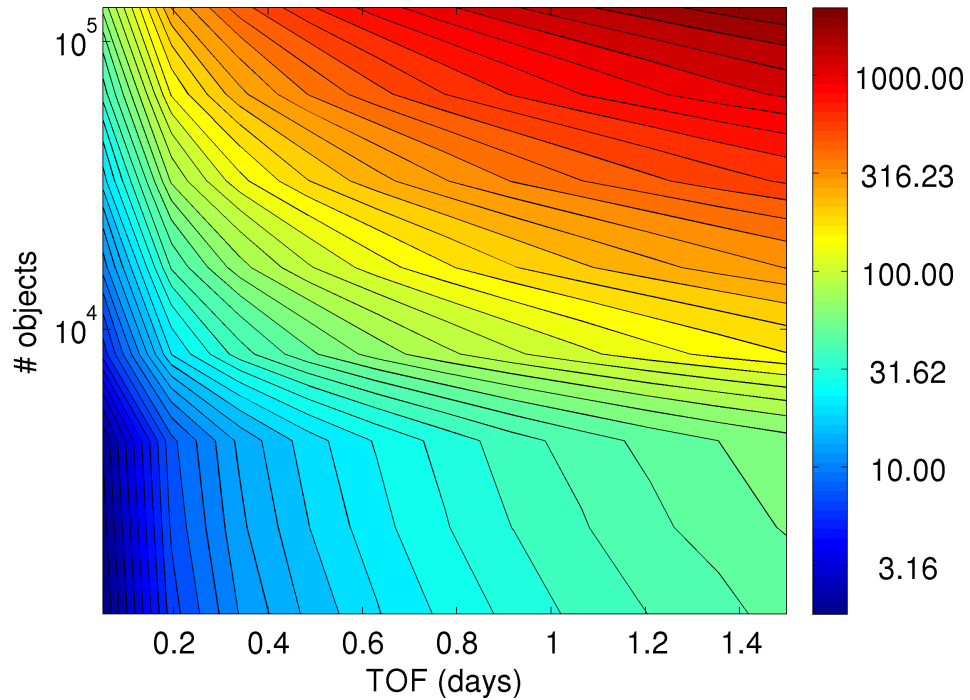
Figure 111: Case 2: absolute runtime (sec), 360×360 SH field

store the state vector at all intermediate integration steps. For both case 1 and case 2 and for SH filed size of 156, simulations demonstrate a runtime of less than an hour
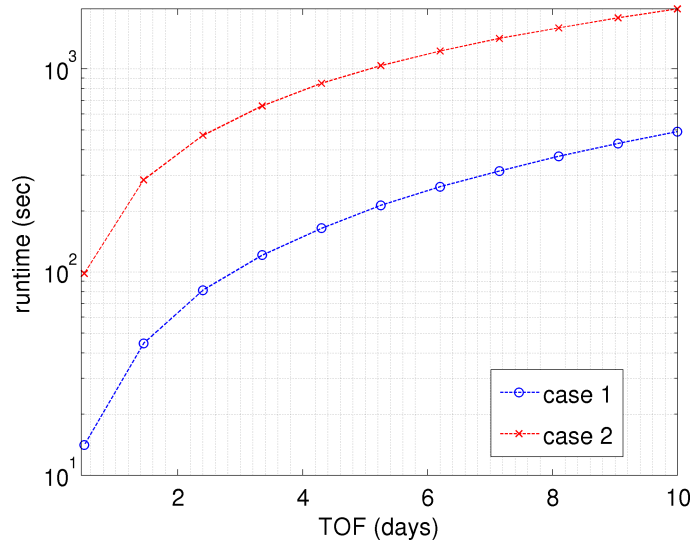
Figure 112: Absolute runtime (sec), Max $TOF = 10$ days, 156×156 SH field, # objects $= 16{,}384$

Table 36: GPU code profile summary (Case 1, # objects $= 16{,}384$)

Field size	L1 local hit rate (%)	L2 throughput (GB/sec)	Local memory overhead (%)	Global load throughput (GB/sec)	IPC	Local memory cache replay overhead (%)
70×70	33.096	63.913	152.881	5.516	0.731	10.838
156×156	37.791	51.354	140.762	4.894	0.793	9.039
360×360	39.464	44.290	129.483	4.883	0.816	8.155

when integrating 100,000+ space objects for 1 day flight time. Such simulations on the CPU (using SH and SPICE) would take weeks or months to finish. Note that the GPU runtime actually decreases when the Fetch model SH field size is increased. To better understand this behavior the GPU code was profiled and the results are summarized in Table 36.

As shown in Table 36, there is a significant increase in local memory overhead and a decrease in L1 cache hit rate with a decrease in Fetch model SH field size. This behavior can be explained from that fact that low order Fetch models have cells which occupy a larger volume in space. Hence, during coefficient lookup, threads from multiple wraps (a group of 32 GPU threads) are forced to read from the same memory location more frequently. Unlike a CPU cache, the GPU cache performs poorly if multiple threads from different wraps read from the same memory location. This taxing of the GPU hardware also reduces the number of instructions issued per

168

GPU clock cycle (IPC), thereby further decreasing the GPU runtime performance.

Figure 112 gives the absolute runtime performance of the tool when simulating 16,384 space objects, using the 156×156 SH gravity field, for long flight times. Specifically, for a flight time of 10 days the tool takes 8.20 and 32.98 minutes (for cases 1 and 2) to simulate 16,384 objects. The performance analysis demonstrates that the methodology is capable of performing high-fidelity integration of multiple objects at unprecedented speeds (on a single workstation), and is applicable to a wide range of problems ranging from conjunction prediction to Monte Carlo analysis to particle filters.

6.9 Chapter Conclusion

In this chapter, a fast high-fidelity multi-spacecraft trajectory integration tool is demonstrated by incorporating recently developed fast gravity perturbation models and by utilizing the modern GPU architecture. The tool takes advantage of the fast and accurate interpolation models developed in chapters 3 and 4 and a GPU solver to achieve massive parallelism across multiple spacecraft. For integration the tool uses a variable step GPU based Runge Kutta integrator to achieve parallelism across multiple threads, while each thread is allowed to enjoy the speedup obtained from using the fast perturbation models. Combining these two approaches leads to multiplicative speedups when compared to simulations in serial on a single CPU. In its current form the tool demonstrates sustained three to four orders of magnitude in speedups for a range of typical flight times with 10,000 objects. While for state-of-the art space-catalog applications the tool achieves two to three orders of magnitude in speedups. Long time of flight simulations for 16,384 objects are also demonstrated. The methodology and the tool has application in a variety of space surveillance applications including: the conjunction problem, covariance realism, particle filters, constellation design and Monte Carlo analyses.

CHAPTER VII

CONCLUSION

7.1 Dissertation Summary

In this thesis, five high-impact astrodynamics problems are identified and their state-of-the art algorithms are systematically improved. Theoretical and methodological improvements are combined with modern computational techniques, resulting in increased algorithm robustness and faster runtime performance.

The multiple revolution Lambert problem is chosen as the first problem for improvement. As this is an important problem in preliminary mission design and analysis, a new universal variable Lambert formulation based on a cosine transformation is described in chapter 2. The formulation is straight-forward and benefits from a robust solution procedure driven by a technique to generate accurate initial guesses. It has similar accuracy as the current state-of-the art Gooding's method and results in 40% to 60% reduction in runtime.

Chapters 3 and 4 focus on improving the runtime performance of two high-fidelity perturbation models. In chapter 3 a global 3D interpolation model called Fetch is developed and applied to interpolate the GRACE GGM03C Earth gravity model. Various numerical and methodological innovations are combined to achieve continuity, non-singularity, adaptivity, global resolution and speed. Four Fetch models of varying resolutions are generated. The memory footprint of these four models varies between 120 MB to 2.5 GB, while the speedup (over the Pines spherical harmonics implementation) varies between 10x to 3,900x. In chapter 4, a simpler 1D interpolation is implemented to construct a new ephemeris computation system called FIRE (Fast

Interpolated Runtime Ephemeris). FIRE is designed for custom trajectory applications and uses spline interpolation along with a multi-level computation architecture to achieve an order of magnitude in speedup over JPL's SPICE ephemeris system. A major benefit of FIRE is its ability to provide smooth, accurate and self consistent derivatives.

In chapter 5, a novel methodology is introduced for parallel sensitivity computation using both the CPU and GPU in tandem for a single trajectory. A tool based on the methodology computes first order sensitivities at almost no extra computational cost when compared to integrating just the state on the CPU. Second order sensitivities are computed with order of magnitude speedups, for example two-body integration.

Finally, in chapter 6, the Fetch and FIRE perturbation models are implemented on the GPU and combined with a GPU based integrator to compute multiple high-fidelity spacecraft trajectories simultaneously on a single workstation at unprecedented speeds. The solution approach is tested for up to 131,072 objects in low to medium Earth orbit with a flight time of up to 10 days. Two to four orders in magnitude speedups are demonstrated when compared with similar computations on the CPU. Given the increasing number of objects in space, the tool provides promising capabilities and is relevant to conjunction analysis problems, covariance realism, particle filters and Monte-Carlo analyses.

7.2   Recommendations For Future Work

The work done in this thesis can be extended and applied to a large variety of problems. In this section recommendations for possible future work are given for the five problems solved in the thesis. Possible, future high-impact problems in astrodynamics are also identified.

7.2.1 Multiple revolution Lambert problem

- The Lambert algorithm developed in the thesis uses a second order root finding method. An initial investigation shows that a higher order root finding maybe faster and is worth investigating further.

- Spline fitting the universal variable corresponding to the minimum multiple revolution time of flight as a function of the new geometry based parameter would eliminate the need to compute the minimization root solves.

7.2.2 High-fidelity geopotential computation

- In its current form, the Fetch gravity model fails to satisfy the Laplace equation. Modifying the current models to satisfy the Laplace equation by using alternative candidate fitting functions may be beneficial.

- The Fetch interpolation model can also be applied to other solar system objects such as the moon, other planets, comets, asteroids, etc.

- Applying the Fetch model to generate high-fidelity atmospheric density models can also be studied.

- Inclusion of temporal gravity effects such as those caused by tides may also be investigated.

7.2.3 Ephemeris computation

- FIRE can be extended to include other bodies like comet, asteroids and artificial spacecraft.

- Spline interpolation accuracy may be improved by using Chebyshev interpolation points.

7.2.4 Fast and accurate sensitivity computation

- A possible future work would be to simplify the GPU side derivative computation by using complex-step differentiation.

- The heterogeneous sensitivity computation algorithm is a good candidate for dynamic parallelism and stands to benefit from the new NVIDIA Kepler architecture.

- The proposed algorithm also has potential application in many fields of numerical optimization. Hence, one could adapt and apply the proposed algorithm to other engineering optimization problems like those encountered in Chemical and Electrical engineering.

7.2.5 Multiple spacecraft simulation using GPU Computing and Fast high-fidelity gravity perturbation models

- Increase the order of the GPU integration. Apart from being more accurate, this change would also reduce the number of intermediate steps needed to be stored. A reduction in the number of steps would also reduce the number of memory transactions and may therefore decrease the absolute runtimes.

- Restrict the algorithm memory requirement by breaking up the GPU computation between multiple streams. This could allow the integration of up to 1 million objects without requiring a large amount of memory.

- The fidelity of the tool can be further increased by incorporating high-fidelity shape models for computing atmospheric drag and solar radiation pressure perturbations.

7.2.6 Future high-impact problems in astrodynamics

Given below is a partial list of other high-impact problems in astrodynamics which may be considered as part of future studies:

1. Investigating higher-dimensional (4 or more) volume interpolation schemes for high-fidelity temporally aware atmospheric density modeling.

2. Designing of fast and parallel numerical optimization algorithms.

3. Parallelizing the multi-body tour design problem.

4. Investigating and designing parallel numerical integration schemes.

7.3 *Primary Thesis Contributions*

Table 37 briefly summarizes the main contributions (new, to the author's knowledge) of this thesis.

Table 37: Summary of thesis contributions

Problem	Contribution
Multiple revolution Lambert problem	→ derived a universal Lambert formulation based on a cosine transformation → developed a new geometry parameter considerably reducing the number of minimization calls necessary to solve the multi-rev problem → demonstrated method accurate and 1.75 to 2.15 times faster (on average) than the current state-of-the art Gooding's method
High-fidelity geopotential computation	→ modified Junkins weighting function method to allow for variable node spacing in the radial direction → developed a two level overlapping grid strategy to overcome the singularity at the poles found in spherical coordinates → developed a parallel coefficient generation algorithm utilizing a novel adaptive polynomial selection strategy to optimize model memory footprint → released four Fetch models of field sizes: 33, 70, 156 and 360; demonstrating speed improvements ranging from one to three orders of magnitude over spherical harmonics

Ephemeris computation	→ developed a multilevel ephemeris computation system for trajectory applications that favor speed and smooth derivatives
Fast and accurate sensitivity computations	→ developed a multi-level trajectory decomposition methodology for parallel overlapping sensitivity computation across a single trajectory; leading to first order sensites being computed at almost no extra computational cost than a state integration on the CPU
Multiple spacecraft simulation using high-fidelity gravity models	→ developed a high-fidelity trajectory simulation tool, combining a GPU based integrator and higher order Fetch and FIRE perturbation models

7.4 Global Summary

In this thesis novel attempts have been made to improve the runtime performance of high impact astrodynamics algorithms. An interdisciplinary approach is adopted, merging the field of astrodynamics and high performance computing to improve algorithm robustness and performance.

Most of the algorithms presented in this thesis stand to benefit from future CPU and GPU improvements. CPUs with better branch prediction capabilities and faster transcendental functions should benefit the Lambert algorithm proposed in chapter 2. The Fetch and FIRE models are expected to benefit from the increasing L1, L2 cache size of modern CPUs. As computer memory becomes less expensive and more abundant, the use of high-fidelity models, which trade memory for speed, is becoming increasingly justified. The algorithms presented in chapters 5 and 6 are also expected to benefit from the improving double precision computing performance of future GPUs. However, some minor algorithmic changes may be required in order to tap the full potential of newer GPU architectures.

The work done in this thesis has wide applicability in astrodynamics and other similar fields. Many problems from the preliminary mission design to high-fidelity trajectory simulation, orbit estimation, and optimization stand to benefit. The work presented here is of relevance to both the academic community and to research institutions such as the Air Force Research Laboratory (AFRL) and National Aeronautics

and Space Administration (NASA). The high-fidelity ephemeris and gravity models would help AFRL effectively tackle challenging problems like conjunction analysis, non-linear filtering and high-fidelity tracking of the space catalog objects. Furthermore, lessons learned from the NASA Dawn mission and the recently proposed NASA asteroid mission emphasize the importance of the high-fidelity gravity modeling of small bodies. The Fetch model could be adapted and applied to model these complex gravity environments. Apart from gravity modeling, the Fetch model can also be used to model Earth's atmospheric density and hence be of value to the space and environmental science community. The field of numerical optimization can utilize and benefit from the parallel sensitivity computation methodology presented in this thesis. The methodology has application in various engineering fields ranging from aerospace engineering to chemical engineering to electrical engineering.

Finally, a multidisciplinary approach is utilized in this thesis to tackle challenging computational problems in astrodynamics. The techniques are found to be effective for the problems considered, combining theoretical and methodological improvements with modern computational hardware and numerical methods.

APPENDIX A

ALGORITHMS

Algorithm 1 Candidate polynomial set generation

1: **procedure** POLYGET(O_{max})   ▷ Input highest degree of the candidate polynomial ≥ 2
2:    $i \leftarrow 0$
3:    $n(1) \leftarrow 1$
4:    **for** $j = 2$ **to** O_{max} **do**   ▷ Major polynomial loop
5:       $n(j) \leftarrow n(j-1) + j$ ▷ Calculate number of terms to be dropped (Eq. 65)
6:       **for** $k = n(j) - 1$ **to** 0 **do**   ▷ Minor polynomial loop
7:          $i \leftarrow i + 1$
8:          $C_{(N-k)\ldots N} \leftarrow 0$ ▷ Set $(N - k)$ to N coefficients to zero
9:          $P_i \leftarrow Q_{j,k}$ ▷ Compute minor polynomial with last $k + 1$ coefficients set to zero
10:      **end for**
11:      $i \leftarrow i + 1$
12:      $P_i \leftarrow Q_j$ ▷ Compute major polynomial with N coefficients
13:   **end for**
14:   **return** $P_{1\ldots i}$ ▷ The final set of candidate polynomials
15: **end procedure**

Algorithm 2 TR aglorithm for block size of 8

1: **procedure** TR ▷ Suppose we have a total of 8 threads, hence 8 first order state transition matrices multiply in descending order ▷ load matrices to each thread in the following order
2:    $thread_1 = M_7,\ thread_2 = M_3,\ thread_3 = M_5,\ thread_4 = M_1$
3:    $thread_5 = M_6,\ thread_6 = M_4,\ thread_7 = M_0$ ▷ this loading can be automated using bit-wise operations; next we do recursive multiplication, each matrix multiplication is done is parallel at each iteration and uses shared memory to enable data reuse   ▷ 1st iteration
4:    $thread_1 : M_{76} = M_7 * M_6,\ thread_2 : M_{32} = M_3 * M_2,\ thread_3 : M_{54} = M_5 * M_4 ;,\ thread_4 : M_{10} = M_1 * M_0;$
5:    $threads\ synchronize$ ▷ 2nd iteration
6:    $thread_1 : M_{7654} = M_{76} * M_{54},\ thread2 : M_{3210} = M_{32} * M_{10}$
7:    $threads\ synchronize$ ▷ 3rd iteration
8:    $thread_1 : M = M_{7654} * M_{3210}$
9:    **for** $threadID == 1$ **do**   ▷ the final thread writes the results back to the GPU global memory, other threads stall
10:      $globalMem(thread_{ID}) = M$
11:   **end for**
12: **end procedure**

Algorithm 3 General master worker parallel algorithm

1: **procedure** MASTER-WORKER(N_{procs})   ▷ Input the number of processors N_{procs}
2:     Initialize all MPI tags   ▷ Used to communicate between master and worker threads
3:     Allocate work arrays per thread
4:     **if** Thread I.D = master **then**   ▷ Master thread part
5:        Generate initial $work$ for each worker thread
6:        $MPI\_SEND(work, 3, MPI\_INTEGER, worker_i, worktag, MPI\_COMM\_WORLD, flag)$ ▷ Send work to each worker thread
7:        **for** $i = 1$ **to** $Total\ Work\ Size$ **do**
8:          $MPI\_RECV(size, 1, MPI\_INTEGER, MPI\_ANY\_SOURCE, donetag, MPI\_COMM\_WORLD, status, flag)$ ▷ Notification from each thread
9:          $sender \leftarrow status(MPI\_SOURCE)$ ▷ Check who sent the notification
10:          **if** $count\ <\ Total\ Work\ Size$ **then**   ▷ Make sure if we need to assign new work
11:            $count \leftarrow count + 1$
12:            Allocate new work for each thread
13:            $MPI\_SEND(work, 3, MPI\_INTEGER, sender, worktag, MPI\_COMM\_WORLD, flag)$ ▷ Send work to each worker thread
14:            update global work counter
15:          **else**
16:            $MPI\_SEND(1, 1, MPI\_INTEGER, sender, quittag, MPI\_COMM\_WORLD, flag)$ ▷ quit flag initiated
17:          **end if**
18:        **end for**
19:     **else**   ▷ Worker thread part
20:        $200 \leftarrow CONTINUE$
21:        $MPI\_RECV(work, 3, MPI\_DOUBLE\_PRECISION, master, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, status, ierr)$ ▷ Receive work from master
22:        **if** $status(MPI\_TAG)$ = quittag **then**
23:          CONTINUE
24:        **else**
25:          Do work
26:          Store work   ▷ Each worker threads stores its own work
27:          $count_{worker} \leftarrow count_{worker} + 1$ ▷ Keep track of how much work each thread is doing
28:          $MPI_SEND(count_{worker}, 1, MPI\_INTEGER, master, donetag, MPI\_COMM\_WORLD, ierr)$
29:          $GOTO \leftarrow 200$
30:        **end if**
31:     **end if**
32: **end procedure**

RADIAL ACCELERATION: GGM03C GRAVITY MODEL
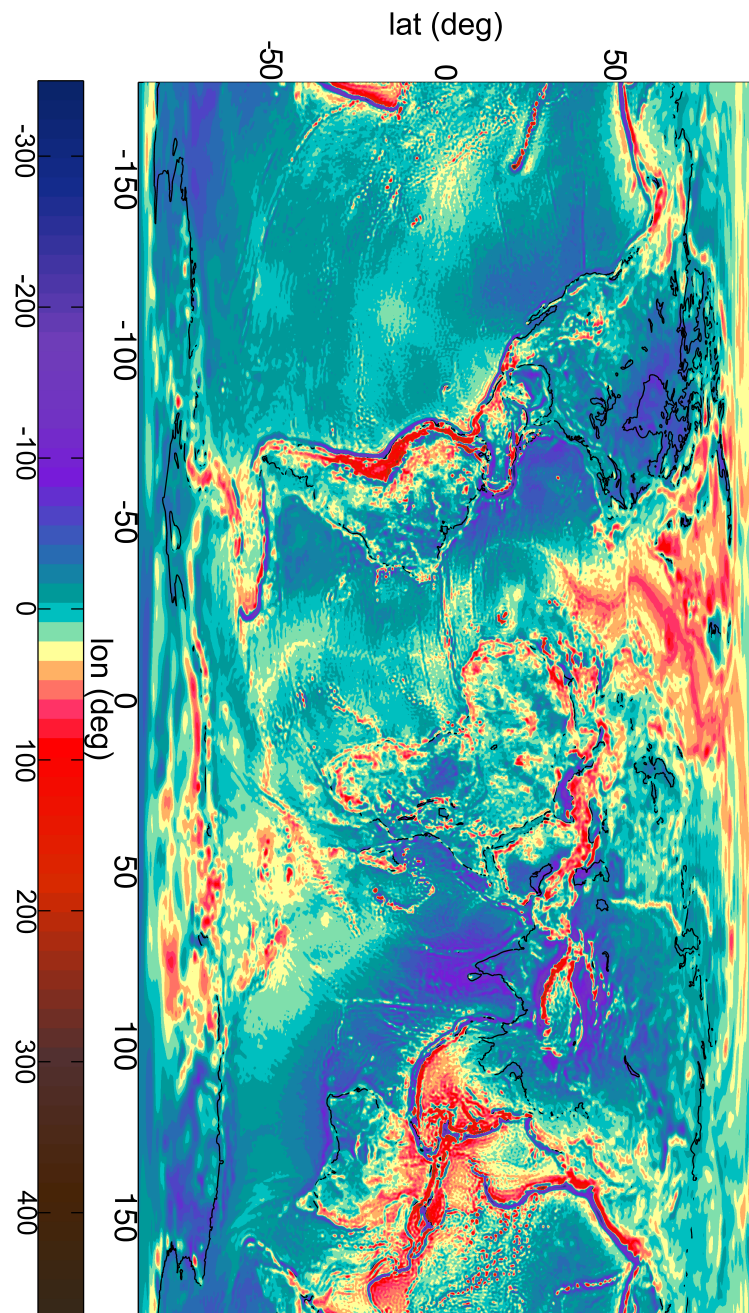
Figure 113: Radial acceleration (in mGals) for 360×360 GGM03C field at surface (two body and J_2 terms removed)

APPENDIX C

PUBLICATION HISTORY

C.1 Relevant Journal Publications

C.1.1 Published

1. Arora, N. and Russell, R. P., "A Fast, Accurate, and Smooth Planetary Ephemeris Retrieval System", Celestial Mechanics and Dynamical Astronomy, Vol 108, No 2, Pp 107-124 (DOI:10.1007/s10569-010-9296-0).

C.1.2 Under review

1. Arora, N. and Russell, R. P.,"High-Fidelity Geopotential Interpolation: Application to the Grace GGMO3C Gravity Model", submitted to the AIAA Journal of Guidance Control and Dynamics

C.1.3 To be submitted

1. Arora, N. and Russell, R. P.,"A Fast and Robust Multiple Revolution Lambert Algorithm using a Cosine Transformation", will be submitted to the AIAA Journal of Guidance Control and Dynamics.

2. Arora, N. and Russell, R. P.,"Fast Sensitivity Computation for Trajectory Optimization", will be submitted to the AIAA Journal of Guidance Control and Dynamics.

3. Arora, N. , Russell, R. P. and Vivek Vittaldev,"Multiple High-Fidelity Space Trajectories using GPU and Fast Perturbation Models", will be submitted to journal Celestial Mechanics and Dynamical Astronomy.

C.2 Relevant Conference Publications

1. Arora, N. and Russell, R. P., "High-Fidelity Geopotential Interpolation: Application to the Grace GGMO3C Gravity Model", Paper AAS 13-321, AAS/AIAA Astrodynamics Specialist Conference, Kauai, HA, Feb 2013.

2. Arora, N. and Russell, R. P., Fast, "High-Fidelity, Multi-Spacecraft Trajectory Simulation for Space Catalogue Applications", Paper S1.3, US-China Space Surveillance Technical Interchange, Beijing, China, Oct. 17-21, 2011.

3. Arora, N. and Russell, R. P., "Fast, Efficient and Adaptive Interpolation of the Geopotential". Paper AAS 11-501, AAS/AIAA Astrodynamics Specialist Conference, Girdwood, AK, Alaska, Aug 2011.

4. Arora, N. and Russell, R. P., "A GPU Accelerated Multiple Revolution Lambert Solver for Fast Mission Design", Paper AAS 10-198, AAS/AIAA Space Flight Mechanics Meeting, San Diego, CA, Feb 2010.

5. Arora, N. and Russell, R. P., Vuduc R., "Fast Sensitivity Computations for Trajectory Optimization", Paper AAS 09-337, AAS/AIAA Astrodynamics Specialist Conference and Exhibit, Pittsburgh, PA, Aug 2009.

6. Russell, R. P. and Arora, N., "FIRE: A Fast, Accurate, and Smooth Planetary Body Ephemeris Interpolation System", Paper AIAA-2008-6278, AAS/AIAA Astrodynamics Specialist Conference and Exhibit, Honolulu, HI, Aug 2008.

C.3 Other Publications

C.3.1 Journal

1. Russell, R. P. and Arora, N., "Global Point Mascon Models for Simple, Accurate and Parallel Geopotential Computation", Journal of Guidance, Control, and Dynamics, Vol. 35, No. 5, 2012, pp. 1568-1581, DOI: 10.2514/1.54533

2. Dutta, A., Arora, N. and Russell, R. P.,"A Peer-to-Peer Refueling Strategy using Low Thrust Propulsion", Journal of Spacecraft and Rockets, (accepted and revised Feb 2012).

C.3.2 Conference

1. Vittaldev, V., Russell, R. P., Arora, N. and Gaylor, D.,"Second Order Kalman Filter Using Multicomplex Step Derivatives", Paper AAS 12-204, AAS/AIAA Space Flight Mechanics Meeting, Charleston, SC, Jan 2012.

2. Russell, R. P., Arora, N.,"Global Point Mascon Models for Simple, Accurate, and Parallel Geopotential Computation", Paper AAS 11-158, AAS/AIAA Space Flight Mechanics Meeting, New Orleans, LA, Feb 2011.

3. Arora, N., Shringarpure A. and Vuduc R.,"Direct n-body kernels for multicore platforms", In Proc. Int'l. Conf. Parallel Processing (ICPP), Vienna, Austria, September 2009.

4. Dutta, A., Arora, N. and Russell, R. P.,"A Peer-to-Peer Refueling Strategy using Low Thrust Propulsion", AAS 09-392, AAS/AIAA Astrodynamics Specialist Conference and Exhibit, Pittsburg, PA, Aug 2009.

C.4 Book Chapters

1. Kang, S., Arora, N., Shringarpure A., Vuduc R. and A. Bader, D,"On the Evaluation of Current Multicore Processors and Accelerator Architectures for Dense Numerical Computations", Handbook of Multi and Many-Core Technologies, (in Press)

APPENDIX D

CODE SETUP AND IMPLEMENTATION

In this appendix, the code setup and usage details for some of the algorithms and methods presented in this thesis is briefly stated. The CPU based codes are written in Fortran and have been tested with the Intel Fortran complier on Linux. A brief summary on GPU programming and Fortran-GPU code setup (used in chapters 5 and 6) is also provided.

D.1 The Universal Variable "k" Based Lambert Formulation

The general setup cost of the proposed Lambert implementation is similar to that of other Lambert solvers found in the literature. The code consist of a single Fortran module (called "KLAM_MOD") which is distributed via the "Klam.f90" Fortran source file. For using the code, a user needs to be familiar with the standard Lambert problem terminology (briefly explained in the code comments section). The user interfaces with the "K_Lam" module via the main Fortran subroutine (called "get_klam"). For a given set of inputs, the subroutine outputs the velocity vectors along with error codes and other relevant solution information (explained in the code comments section). A tuned Lambert source code is presented as a part of this thesis in appendix E.

Given the simplified form of the Lambert time of flight equation and its derivatives, implementing the basic algorithm is straightforward. On the other hand, implementing the initial guess strategy is complex and may significantly increase the code setup and implementation time. Nevertheless, all the details for successfully implementing the initial guess are provided in chapter 2.

Table 38: Fetch model implementation details

Phase	Number of files	Computation type	Required supporting data
Model coefficient generation	8	Parallel via MPI	Maple inverse matrices data files
Fetch runtime routines	1	Serial (single CPU thread)	Model coefficient file

D.2   High-Fidelity Geopotential Computation: The Fetch model

The Fetch model is implemented in two phases. The first phase is responsible for generating model coefficients using a MPI based parallel algorithm. The second phase is responsible for computing the geopotential and its higher order derivatives using the Fetch runtime routines. Table 38 briefly summarizes the code setup for both the phases.

The coefficient generation routines interface with the user via a namelist input file. The user is responsible for selecting a value of the parameter Γ (see Eq. 77), the tolerance multipliers $\eta_{0...4}$ (see Eq. 76) and the overlapping latitude. The coefficient generation routines also rely on data files containing analytical inverse matrices data which are computed via a Maple worksheet. The result of the coefficient generation phase is a single Fetch model coefficient file for a given SH degree and order.

Similar to spherical harmonics (SH), using the Fetch runtime code is a two step process, as given below:

1. Initialize the Fetch model and load the coefficient file (analogous to loading the SH coefficients)

2. Call the "get_Fetch" or "get_FetchEZ" runtime subroutines with appropriate derivative flags

The runtime subroutines are made accessible to the user through the "FETCH_MOD" module which is distributed via the "Fetch.f90" source file. The setup cost for using the Fetch model is similar to that of SH, with the exception being that it requires more memory (random access memory) for loading the Fetch model coefficients at runtime.

The loading of the coefficient file (a one time process) can take from 2 seconds to 20 seconds depending upon the operating system and the memory performance of the workstation.

Implementing the Fetch runtime algorithm from scratch requires a detailed understanding of the modified Junkins weighting function method. To overcome this complexity, the runtime source code is provided as a part of this thesis in appendix E.

D.3 Ephemeris Computation: The FIRE system

As stated in chapter 4, FIRE is written in the Fortran programming language and follows a modular, multilevel implementation strategy. The main architecture is divided into three subsystems:

1. Archived Cubic spline Ephemeris (ACE); typically computed only once

2. Runtime Adaptive Custom Ephemeris (RACE); custom computed for a class of problems

3. RACE loading and Runtime batch processing routines

FIRE currently uses the JPL's SPICE ".bsp" and ".tpc" files[2] to generate the archived cubic spline ephemeris (ACE), although FIRE could easily be tailored to use any established ephemeris as input. Each subsystem contains its own set of local core routines. A standard namelist based input method is followed for each of the subsystems and imparts robustness and flexibility to the code. SPICE naming convention is adopted for the custom tree generation to preserve consistency.

The user typically computes the ACE once from an underlying base ephemeris. In this thesis, SPICE is used as the base ephemeris system. Once ACE is created, the user can create multiple custom RACE's for different types of problems. Finally,

using the runtime routines the user loads a RACE file and computes the required state and orientation data.

FIRE, has in total nine different runtime subroutines (shown in in Table 39) which can be called at runtime using the Fortran module "MAKE_CFIRE". Five of these subroutines give states (xyz positions and uvw velocities) and orientations while the remaining four additionally provide derivatives. Various combinations of "P, V, R and D" are used to call for appropriate data type. For example, if only position and rotation data is needed then the user would call the routine "FIRE_PR". This provides a user friendly interface that is consistent with the existing functionality of SPICE.

Table 39: FIRE runtime routines and their function

Routine name	Function performed:
FIRE_P	Computes xyz states
FIRE_PV	Computes xyz and uvw states
FIRE_PVR	Computes xyz, uvw states plus the rotation matrix
FIRE_R	Computes the rotation matrix
FIRE_PR	Computes xyz and the rotation matrix
FIRE_PD	Computes xyz states, and corresponding derivatives
FIRE_PVD	Computes xyz, uvw states, and corresponding derivatives
FIRE_PVRD	Computes xyz, uvw states, the rotation matrix, and corresponding derivatives
FIRE_DR	Computes the rotation matrix, and corresponding derivatives

D.4   GPU Based Parallelism

D.4.1   NVIDIA GPU programming

Recent advances in the programmable GPU has lead to the development of a highly parallel and multi-threaded processor with many-cores. Given the GPU's high computational power and its ability to tap fine grain parallelism, researchers are now mapping non-graphical applications to the hardware with a wide range of success.[15,7,64,108,135,5] This field is generally called GPGPU (General Purpose Computing on GPU's) programming. The main breakthroughs in GPGPU programming came with the development of NVIDIA TESLA and FERMI architectures (in late 2006,[82] and 2010[143])

along with the introduction NVIDIA CUDA [1] technology. Before CUDA, advanced GPU programming knowledge was required to exploit the hardware effectively and it was still not very efficient. Post CUDA, there has been tremendous growth in wide scale GPGPU programming applications on the TESLA processors. Most of these applications have witnessed a performance boost of 5 to 500 times, thereby outperforming many mid-range supercomputers. With the addition of double precision floating point arithmetic support to CUDA, it is possible now to achieve performance increase without sacrificing accuracy. The main task is to design an algorithm which maps well to the GPU and exploits this abundant computing power.

D.4.2 CUDA (Compute Unified Device Architecture)

The CUDA computing architecture is a C-like programming language with keywords for labelling data-parallel functions (kernels), and their associated data structures. Kernels generally execute a large number of threads (on the order of tens of thousands) in parallel. A thread is basically a fork which results from concurrent execution of computation on the GPU. Typically, in the GPU programming model, thousands of threads perform the same set of operations over a different set of data. It is worth noting that CUDA threads are computationally lighter than the threads on the CPU and hence they need very few cycles to generate and schedule.

The NVIDIA C Compiler (NVCC) is responsible for compiling the CUDA code. The part of the code which runs on the GPU is called the device code and the part of the code running on the CPU is called the host code. The host and device codes can be compiled using different compilers and linked at runtime.

The GPU execution starts with the host invoking a kernel function, where a large number of threads are spawned. All threads which run on a kernel are collectively called a grid block. This gird block is further divided into smaller units called thread

---

[1]http://www.nvidia.com/object/cuda_what_is.html

blocks. Each thread block can have at most 1024 threads (512 on old GPU architectures), which can communicate and synchronize among each other via shared memory (up to 48 KB per thread block on new Fermi architecture). When all threads of a kernel complete their execution, the corresponding grid terminates and the execution continues on the host code until another kernel is invoked. Multiple kernels may also be launched simultaneously using streams if the GPU resources are free. The general, main points which have to be kept in mind while designing a CUDA algorithm are as follows:

- designing a fine-grained parallel algorithm with sufficient amount of independent thread blocks to hide global memory latency (time to access GPU's global memory)

- using shared memory for data reuse within a thread block (according to NVIDIA shared memory is 300 times faster than the global memory)

- coalesced and conflict free memory access between multiple memory abstractions (device memory, shared memory, register memory)

- minimizing and/or hiding CPU-GPU memory transfers (PCI Bus transfers) as they are slow and hence directly affect the performance

- optimizing register usage which restricts the number of threads and thread blocks which can be deployed simultaneously

- concurrent execution (overlapping work between CPU and GPU) and stream computing

All of these topics make it challenging to develop algorithms which map effectively to the GPU. Often non-intuitive techniques are developed to map conventionally serial algorithm to the GPU [1]. Once an algorithm is developed, achieving high performance (5x-50x) is possible. Very high performance boost (up to 100x or more) are possible

---

[1]http://developer.download.nvidia.com/compute/cuda/sdk/website/samples.html#scan

only if the algorithm maps efficiently to the GPU hardware and a sufficient amount optimization has been performed. Hence, algorithm development is the major activity for consideration when programming in CUDA. Fig. 114 gives an overview of the CUDA programming model.
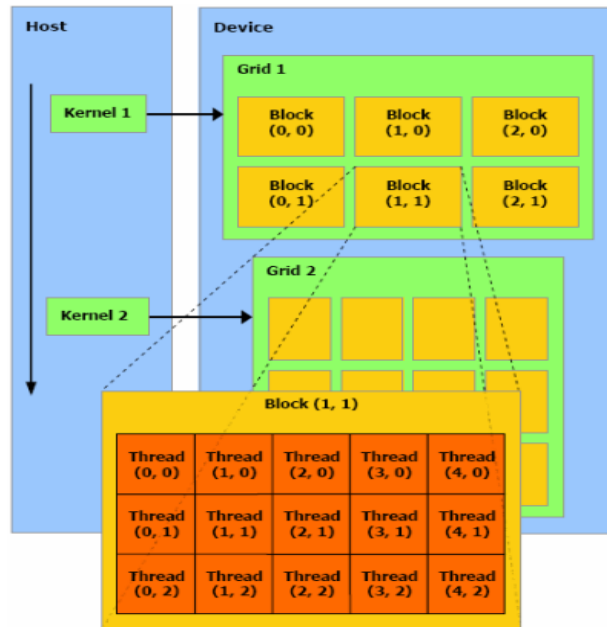
Figure 114: CUDA programming model [figure taken from[91]]

To make a computer system capable of running GPU based algorithms, a GPU which supports CUDA (check NVIDIA CUDA programming guide for more information), the latest NVIDIA CUDA drivers and NVIDIA CUDA toolkit [1] is required. Once installed, the CUDA and its accompanying C code can be complied using NVIDIA's NVCC compiler.

D.4.3 Fortran-CUDA interface

Figure 115 outlines the general GPU interface strategy followed in chapters 5 and 6. The Fortran based CPU code communicates with C/CUDA code functions and libraries using the ISO C Bindings feature present in the Fortran 2003. An ISO

---

[1]https://developer.nvidia.com/cuda-downloads
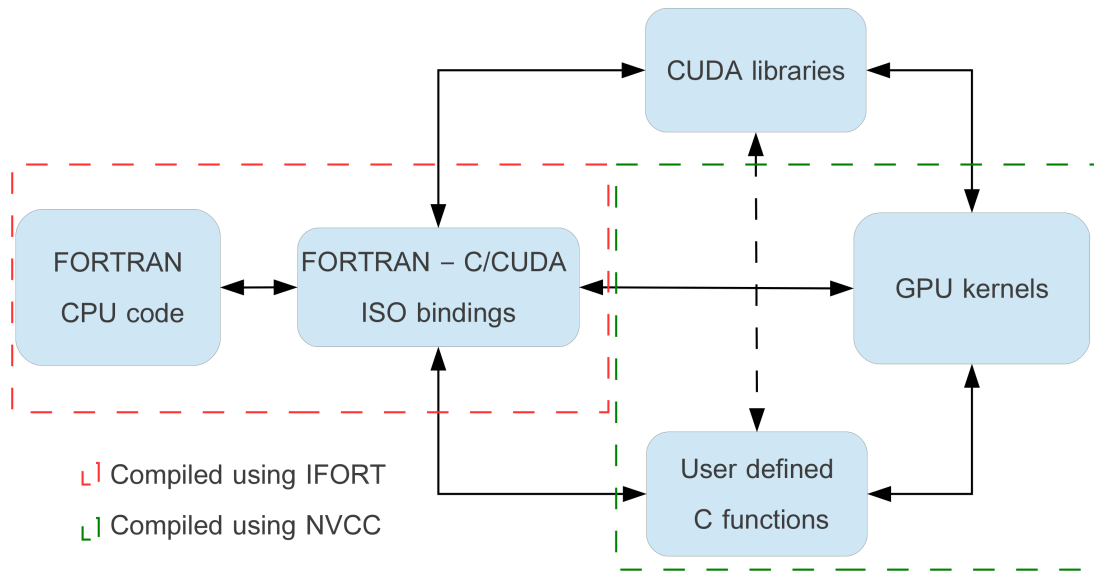
Figure 115: General GPU code workflow

binding based wrapper for the most commonly used functions in CUDA was created as part of this thesis and is used to control all communication between Fortran and C/CUDA.

APPENDIX E

EMBEDDED CODES

E.1 Multiple Revolution Lambert Solver Using The Universal Variable "k"

The universal variable "k" based Lambert solver source code can be obtained by double clicking: . A user may need to rename the attached file to "klam.f90" in some cases. A more up to date version of runtime code can be found here:

http://russell.ae.utexas.edu/index_files/lambert.htm

E.2 The Fetch Model Runtime Routines

The runtime Fetch routines source code can be obtained by double clicking: . As with the previous code, a user may need to rename the attached file to "Fetch.f90" in some cases.

The coefficient files of the four Fetch models and an up to date version of runtime code can be found here:

http://russell.ae.utexas.edu/index_files/fetch.htm

REFERENCES

[1] ABDELKHALIK, O. and MORTARI, D., "N-impulse orbit transfer using genetic algorithms," *Journal of Spacecraft and Rockets*, vol. 44, 2007.

[2] ACTON, C., "Ancillary data services of nasa's navigation and ancillary information facility," *Planetary and Space Science*, vol. 44, pp. 65–70, 1996.

[3] AIELLO, J., "Numerical investigation of mapping orbits about jupiters icy moons," in *AAS/AIAA Astrodynamics Specialist Conference*, (Lake Tahoe, California), August 2005.

[4] ARORA, N., RUSSELL, R. P., and VUDUC, R. W., "Fast sensitivity computations for trajectory optimization," *AAS/AIAA Astrodynamics Specialist Conference and Exhibit*, 2009.

[5] ARORA, N. and RUSSELL, R. P., "A gpu accelerated multiple revolution lambert solver for fast mission design," *AAS/AIAA Astrodynamics Specialist Conference*, vol. AAS 10-198, 2010.

[6] ARORA, N. and RUSSELL, R. P., "High-fidelity geopotential interpolation: Application to the grace ggmo3c gravity model," *AAS/AIAA Astrodynamics Specialist Conference*, vol. AAS 13-321, 2013.

[7] ARORA, N., SHRINGARPURE, A., and R.W, V., "Direct n-body kernels for multicore platforms," 2009.

[8] ARORA, N. and RUSSELL, R. P., "A fast, accurate, and smooth planetary ephemeris retrieval system," *Celestial Mechanics and Dynamical Astronomy*, vol. 108, pp. 107–124, 2010.

[9] AVANZINI, G., "A simple lambert algorithm," *Journal of guidance, control, and dynamics*, vol. 31, no. 6, pp. 1587–1594, 2008.

[10] BARRIO, R., PALACIOS, M., and ELIPE, A., "Chebyshev collocation methods for fast orbit determination," *Applied Mathematics and Computation*, vol. 99, no. 2-3, pp. 195 – 207, 1999.

[11] BATE, R. R., MUELLER, D. D., and WHITE, J. E., *Fundamentals of astrodynamics*. Dover, 1971.

[12] BATTIN, R. H., "Lambert's problem revisited," *AIAA Journal*, vol. 15, no. 5, pp. 707–713, 1977.

[13] BATTIN, R. H., *An Introduction to the Mathematics and Methods of Astrodynamics*. AIAA Education Series, 1999.

[14] BELEY, K. B., "Analytic determination of perigee passage using lambert's therorem," *Journal of Spacecraft and Rockets*, vol. 4, no. 8, pp. 1101–1103, 1967.

[15] BELLEMAN, R. G., BEDORF, J., and ZWART, S. F. P., "High performance direct gravitational n-body simulations on graphics processing units II: An implementation in CUDA," *New Astronomy*, 2008.

[16] BERRY, M., "Integration of orbit trajectories in the presence of multiple full gravitational fields," *AAS/AIAA Spaceflight Mechanics Meeting*, vol. AAS 08-235, 2008.

[17] BERRY, M. M., *A Variable-Step Double-Integration Multi-Step Integrator*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 2004.

[18] BETTS, J. T., "Survey of numerical methods for trajectory optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, p. 193, 1998.

[19] BEYLKIN, G. and CRAMER, R., "Toward multiresolution estimation and efficient representation of gravitational fields," *Celestial Mechanics and Dynamical Astronomy*, vol. 84, no. 1, pp. 87–104, 2002.

[20] BIEDRON, R., SAMAREH, J., and GREEN, L., "Parallel computation of sensitivity derivatives with application to aerodynamic optimization of a wing," *Computational Aerosciences Workshop (NASA Ames Research Center)*, 1998.

[21] BISCHOF, C., GREEN, L., HAIGLER, K., and T.L. KNAUFF, J., "Parallel calculation of sensitivity derivatives for aircraft design using automatic differentiation," *5th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization Conference*, 1994.

[22] BRYSON, A. E. and HO, Y.-C., *Applied Optimal Control*. 2006.

[23] BURRUS, C. and ESCHENBACHER, P., "An in-place, in-order prime factor fft algorithm," *IEEE Transactions on Computers*, vol. 29, pp. 806–817, 1981.

[24] BYRD, R., SCHNABEL, R., and SHULTZ, G., "Parallel quasi-newton methods for unconstrained optimization," *Journal of Spacecraft and Rockets*, vol. 42, no. 1-3, pp. 273–306, 1988.

[25] CAMPBELL, E. T. and SPECKMAN, L. E., "Preliminary design of feasible athos intercept trajectories," in *Planetary Defense Conference: Protecting Earth from Asteroid*, (Orange County, California), February 23 - 26 2004.

[26] CARRICO, T., CARRICO, J., POLICASTRI, L., and LOUCKS, M., "Investigating orbital debris events using numerical methods with full force model orbit propagation," *AAS Space Flight Mechanics*, vol. AAS 08-126, 2008.

[27] CASOTTO, S. and FANTINO, E., "Evaluation of methods for spherical harmonic synthesis of the gravitational potential and its gradients," *Advances in Space Research*, vol. 40, pp. 69–75, 2007.

[28] COLOMBI, A., HIRANI, A. H., and VILLAC, B. F., "Adaptive gravitational force representation for fast trajectory propagation near small bodies," *Journal of Guidance, Control, and Dynamics*, vol. 31, no. 4, pp. 1041–1051, 2008.

[29] CONFORTI, D., LUCA, L., GRANDINETTI, L., and MUSMANNO, R., "A parallel implementation of automatic differentiation for partially separable functions using pvm," *Parallel Computing*, no. 22, pp. 643–656, 1996.

[30] CONTE, S. and DEBOOR, C. in *Elementary Numerical Analysis*, 1972.

[31] COOLEY, J. W., A. PETER, W., and D.WELCH, P., "The fast fourier transform and its applications," *IEEE Transactions on Education*, vol. 12, pp. 27–34, 1969.

[32] COVERSTONE, V. L. and PRUSSING, J. E., "A class of optimal two-impulse rendezvous using multiple-revolution lambert solutions," *The Journal of the Astronautical Sciences*, vol. 48, no. 2-3, pp. 131–148, 2000.

[33] DAVIS, B., AUYEUNG, M. CLARK, M., LEE, C., THOMAS, M., PALKO, J., and VARNEY, R., "Lessons learned building a general purpose cluster for space mission applications," *Space Mission Challenges for Information Technology*, p. 9, 2006.

[34] DEBOOR, C. D. in *A Practical Guide to Splines*, vol. 27, Springer, 2001.

[35] DESAI, P. N. and CHEATWOOD, F. M., "Entry dispersion analysis for the genesis sample return capsule," in *AAS/AIAA Astrodynamics Specialist Conference*, (Girdwood, Alaska), April 16-19 1999.

[36] Dormand, J. and Prince, P., "A family of embedded runge-kutta formulae," *Journal of Computational and Applied Mathematics*, vol. 6, no. 1, pp. 19 – 26, 1980.

[37] Escobal, P. R., *Methods of Orbit Determination*. John Wiley and Sons, 1965.

[38] et. al. Petropoulos, A., "1st act global trajectory optimisation competition: Results found at the jet propulsion laboratory," *Acta Astronautica*, vol. 61, no. 9, pp. 806–815, 2007.

[39] Fang, D. Z. and Yi-fei, Q., "A new trajectory sensitivity approach for computations of critical parameters," *Electric Power Systems Research*, vol. 77, no. 3-4, pp. 303–307, 2006.

[40] Feehery, W. and Barton, P., "Dynamic optimization with state variable path constraints," *Computers and Chemical Engineering*, vol. 22, no. 9, pp. 1241–1256, 1998.

[41] Fehlberg, E., "Classical fifth, sixth, seventh, and eighth order runge-kutta formulas with stepsize control," tech. rep., NASA, 1968.

[42] Folta, D. C., Woodard, M., Howell, K., and Schlei, C. P. W., "Applications of multi-body dynamical environments: The {ARTEMIS} transfer trajectory design," *Acta Astronautica*, vol. 73, no. 0, pp. 237 – 249, 2012.

[43] for the Assessment of the U.S. Air Force's Astrodynamic Standards; Aeronautics, C., on Engineering, S. E. B. D., and Council, P. S. N. R., *Continuing Kepler's Quest: Assessing Air Force Space Command's Astrodynamics Standards*. The National Academies Press, 2012.

[44] Fukushima, T., "Efficient orbit integration by scaling for kepler energy consistency," *The Astronomical Journal*, vol. 126, no. 2, p. 1097, 2003.

[45] GAUSS, C. F., *Theory of Motion of the Heavenly Bodies Revolving about the Sun in Conic Sections*, vol. 1. Dover, 1963.

[46] GIAMPIERI, G. and DOUGHERTY, M. K., "Rotation rate of saturns interior from magnetic field observations," *Geophysical Research Letters*, vol. 31, 2004.

[47] GILL, P. E., MURRAY, W., and SAUNDERS, M. A., "Snopt: An sqp algorithm for large-scale constrained optimization," *Society for Industrial and Applied Mathematics*, vol. 47, pp. 99–131, 2005.

[48] GOODING, R. H., "A procedure for the solution of lambert's orbital boundary-value problem," *Celestial Mechanics and Dynamical Astronomy*, vol. 48, no. 2, pp. 145–165, 1990.

[49] GUO, Y. and FARQUHAR, R. W., "New horizons mission design for the pluto-kuiper belt mission," vol. AIAA 2002-4722, 2002.

[50] HANS and MUNTHE-KAAS, "High order runge-kutta methods on manifolds," *Applied Numerical Mathematics*, vol. 29, no. 1, pp. 115 – 127, 1999. Proceedings of the NSF/CBMS Regional Conference on Numerical Analysis of Hamiltonian Differential Equations.

[51] HARTWICH, A., STOCKMANN, K., TERBOVEN, C., FEUERRIEGEL, S., and MARQUARDT, W., "Parallel sensitivity analysis for efficient large-scale dynamic optimization," *Optimization and Engineering*, vol. 12, no. 4, pp. 489–508, 2011.

[52] HE, Q., LI, J., and HAN, C., "Multiple-revolution solutions of the transverse-eccentricity-based lambert problem," *Journal of Guidance, Control and Dynamics*, vol. 33, no. 1, 2010.

[53] HEATON, A. F., STRANGE, N. J., LONGUSKI, J. M., and BONFIGLIO, E. P., "Automated design of the europa orbiter tour," *Journal of Spacecraft and Rockets*, vol. 39, no. 1, pp. 17–22, 2002.

[54] HERRICK, S. and LIU, A., "Two body orbit determination from two positions and time of flight," *Appendix A, Aeronutronic*, vol. C-365, 1959.

[55] HOFFMAN, T., "Grail: Gravity mapping the moon," in *Aerospace conference, 2009 IEEE*, pp. 1 –8, march 2009.

[56] HOOTS, F. R., SCHUMACHER, P. W., and GLOVER, R. A., "History of analytical orbit modeling in the u.s. space surveillance system," *Journal of Guidance, Control, and Dynamics*, vol. 27, no. 2, pp. 174–185, 2004.

[57] HOU, H., "The fast hartley transform algorithm," *IEEE Transactions on Computers*, vol. 36, pp. 147–156, 1987.

[58] HUANG, Q., YOKOI, K., KAJITA, S., KANEKO, K., ARAI, H., KOYACHI, N., and TANIE, K., "Planning walking patterns for a biped robot," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 26, 1978.

[59] HUJSAK, R. S., "Gravity acceleration approximation functions," *AAS-96-123*, vol. 93, pp. 335–349, 1996.

[60] HULL, D. G., "Numerical derivatives for parameter optimization," *Journal of Guidance, Control, and Dynamics*, vol. 2, pp. 158–160, 1979.

[61] HWANG, C. and LIN, M.-J., "Fast integration of low orbiter's trajectory perturbed by the earth's non-sphericity," *Journal of Geodesy*, vol. 72, pp. 578–585, 1998.

[62] J., P. C. and LIN, L. C., "Formulation and optimization of cubic polynomial joint trajectories for industrial robots," *IEEE Transactions on Automatic Control*, pp. 508–517, 1983.

[63] JACOBSON, R. A. and FRENCH, R. G., "Orbits and masses of saturns co-orbital and f-ring shepherding satellites," *Icarus*, vol. 172, pp. 382–387, 2004.

[64] JANUSZEWSKI, M. and KOSTUR, M., "Accelerating numerical solution of stochastic differential equations with cuda," *Computational Physics*, 2009.

[65] JEZEWSKI, D. J., "K/s two-point-boundary-value problems," *Celestial Mechanics and Dynamical Astronomy*, vol. 14, no. 1, pp. 105–111, 1976.

[66] JONES, B. A., BORN, G. H., and BEYLKIN, G., "Comparisons of the cubed-sphere gravity model with the spherical harmonics," *Journal of Guidance, Control, and Dynamics*, vol. 33, pp. 415–425, 2.

[67] JUNKINS, J. L., "Investigation of finite-element representations of the geopotential," *AIAA Journal*, vol. 14, no. 6, pp. 803–808, 1976.

[68] JUNKINS, J. L. and ENGELS, R. C., "Local representation of the geopotential by weighted orthonormal polynomials," *Journal of Guidance, Control, and Dynamics*, vol. 3, no. 1, pp. 55–61, 1980.

[69] JUNKINS, J. L., MILLER, G. W., and JANCAITIS, J. R., "A weighting function approach to modeling of irregular surfaces," *Journal of Geophysical Research*, vol. 78, no. 11, pp. 1794–1803, 1973.

[70] KLUMPP, A. R., "New developments in astrodynamics algorithms for autonomous rendezvous," *JPL Document*, no. 19930013090, 1991.

[71] Koch, K. R., "Simple layer model of the geopotential in satellite geodesy," *The Use of Artificial Satellites for Geodesy, Geophysics Monograph Series*, vol. 15, pp. 107–109, 1972.

[72] Koon, W. S., Lo, M. W., Marsden, J. E., and Ross, S. D., "The genesis trajectory and heteroclinic connections," *AAS/AIAA Astrodynamic Specialist Conference*, p. 451, 1999.

[73] Kosti, A., Anastassi, Z., and Simos, T., "An optimized explicit runge-kutta-nystrom method for the numerical solution of orbital and related periodical initial value problems," *Computer Physics Communications*, vol. 183, pp. 470–479, March 2012.

[74] Kriz, J., "A uniform solution of the lambert problem," *Celestial Mechanics and Dynamical Astronomy*, vol. 14, no. 4, pp. 509–513, 1976.

[75] Labunsky, A. V., Papkov, O. V., and Sukhanov, K. G., *Multiple gravity assist interplanetary trajectories*. ESI book series, 1998.

[76] Lam, T. and Hirani, A. N., "Characteristics of transfers to and captures at europa," *Celestial Mechanics and Dynamical Astronomy*, 2006.

[77] Lancaster, E. R. and Blanchard, R. C., "A unified form of lambert's theorem," tech. rep., NASA, 1969.

[78] Lancaster, E. R., Blanchard, R. C., and Devaney, R. A., "A note on lambert's theorem," *Journal of Spacecraft and Rockets*, vol. 3, no. 9, pp. 1436–1438, 1995.

[79] Lantoine, G. and Russell, R. P., "A hybrid differential dynamic programming algorithm for robust low-thrust optimization," *AAS/AIAA Astrodynamics Specialist Conference and Exhibit*, 2008.

[80] LEBRETON, J.-P. and MATSON, D., "The cassini-huygens mission (part i)," *Space Research Today*, vol. 169, no. 0, pp. 11 – 19, 2007.

[81] LEKIEN, F. and MARSDEN, J., "Tricubic interpolation in three dimensions," *International Journal for Numerical Methods in Engineering*, vol. 63, 2005.

[82] LINDHOLM, E., NICKOLLS, J., OBERMAN, S., and MONTRYM, J., "NVIDIA Tesla: A unified graphics and computing architecture," *Micro IEEE*, 2008.

[83] LO, M. W., ANDERSON, R., WHIFFEN, G., and ROMANS, L., "The role of invariant manifolds in low thrust trajectory design," *AAS/AIAA Astrodynamic Specialist Conference*, vol. 288, 2004.

[84] LONGUSKI, J. M. and WILLIAMS, S. N., "Automated design of gravity-assist trajectories to mars and the outer planets," *Celestial Mechanics and Dynamical Astronomy*, vol. 52, pp. 207–220, 1991.

[85] LUNDBERG, J. B. and SCHUTZ, B., "Recursion formulas of legendre functions for use with nonsingular geopotential models," *Journal of Guidance, Control, and Dynamics*, vol. 11, pp. 32–38, 1.

[86] MARSDEN, B. G., "Initial orbit determination - the pragmatist's point of view," *Astronomical Journal*, vol. 90, pp. 1541–1547, 1985.

[87] MATOUSEK, S., "The juno new frontiers mission," *Acta Astronautica*, vol. 61, no. 10, pp. 932 – 939, 2007.

[88] MORRISON, F., "Algorithms for computing the geopotential using a simple density layer," *Journal of Geophysical Research*, vol. 81, no. 26, pp. 4933–4936, 1976.

[89] MOSHIER, S. L., "Self-contained ephemeris calculator," *Astronomy and Numerical Software web site (http://www.moshier.net/)*, 2010.

[90] NELSON, S. L. and ZARCHAN, P., "Alternative approach to the solution of Lambert's problem," *Journal of Guidance Control Dynamics*, vol. 15, pp. 1003–1009, Aug. 1992.

[91] NVIDIA, "Nvidia cuda programming guide 4.0," *Documentation*, 2012.

[92] NYLAND, L., HARRIS, M., and PRINS, J., *Fast N-Body Simulation with CUDA*.

[93] OCHOA, S. I. and PRUSSING, J. E., "Multiple revolution solutions to lambert's problem," *Advances in the Astronautical Sciences*, vol. 79, no. 2, pp. 1989–2008, 1992.

[94] OLTROGGE, D. L., "Astrohd: Astrodynamics modeling with a distinctly digital flavor," in *AAS/AIAA Astrodynamics Specialist Conference,*, (Honolulu, Hawaii), August 18-21 2008.

[95] PARK, R. S. and SCHEERES, D. J., "Nonlinear semi-analytic methods for trajectory estimation," *Journal of Guidance, Control and Dynamics*, vol. 30, pp. 1668–1676, 2007.

[96] PARK, R. S., WERNER, R. A., and BHASKARAN, S., "Estimating small-body gravity field from shape model and navigation data," *Journal of Guidance, Control, and Dynamics*, vol. 33, no. 1, pp. 212–221, 2010.

[97] PATEL, P. and SCHEERES, D. J., "A non-linear optimization algorithm," *AAS*, 2008.

[98] PETERSON, G., CAMPBELL, E. T., BALBAS, J., IVY, S., MERKURJEV, E., HALL, T., and RODRIGUEZ, P., "Relative performance of lambert solvers 1: Zero revolution methods," *Advances in the Astronautical Sciences*, vol. 136, no. 1, pp. 1495–1510, 2010.

[99] PETROPOULOS, A. E., LONGUSKI, J. M., and BONFIGLIO, E. P., "Trajectories to jupiter via gravity assists from venus, earth and mars," *Journal of Spacecraft and Rockets*, vol. 37, nov 2000.

[100] PETROPOULOS, A. E. and RUSSELL, R. P., "Low-thrust transfers using primer vector theory and a second-order penalty method," August 2008.

[101] PINES, S., "Uniform representation of the gravitational potential and its derivatives," *AIAA Journal*, vol. 11, pp. 1508–1511, 1973.

[102] QUINLAN, G. D. and TREMAINE, S., "Symmetric multistep methods for the numerical integration of planetary orbits," *Astronomical Journal*, vol. 100, pp. 1694 – 1700, 1990.

[103] QUINN, T., TREMAINE, S., and DUNCAN, M., "A three million year integration of the earth's orbit," *The Astronomical Journal*, vol. 101, pp. 2287–2305, 1991.

[104] RAO, P., SUTTER, B., and HONG, P., "Six-degree-of-freedom trajectory targeting and optimization for titan launch vehicles," *Journal of Spacecraft and Rockets*, vol. 34, no. 3, pp. 341–346, 1997.

[105] RAYMAN, M. D., FRASCHETTI, T. C., RAYMOND, C. A., and RUSSELL, C. T., "Dawn: A mission in development for exploration of main belt asteroids vesta and ceres," *Acta Astronautica*, vol. 58, no. 11, pp. 605 – 616, 2006.

[106] R.KOCH, K. and MORRISON, F., "A simple layer model of the geopotential from a combination of satellite and gravity data," *Journal of Geophysical Research*, vol. 75, no. 8, pp. 1483–1492, 1970.

[107] R.Koch, K. and Witte, B. U., "Earth's gravity field represented by a simple layer potential from doppler tracking of satellites," *Journal of Geophysical Research*, vol. 76, no. 35, pp. 8471–8479, 1971.

[108] Robilliard, D., Marion, V., and Fonlupt, C., "High performance genetic programming on gpu," *International Conference on Autonomic Computing*, 2009.

[109] Russell, R. P., *Global Search and Optimization for Free-Return Earth-Mars Cyclers*. PhD thesis, The University of Texas at Austin, august 2004.

[110] Russell, R. P., "Primer vector theory applied to global low-thrust trade studies," *Journal of the Guidance,Control and Dynamics*, vol. 30, pp. 460–472, 2007.

[111] Russell, R. P. and Arora, N., "Fire: A fast, accurate, and smooth planetary body ephemeris interpolation system," *AAS/AIAA Astrodynamics Specialist Conference and Exhibit*, 2008.

[112] Russell, R. P. and Arora, N., "Global point mascon models for simple, accurate, and parallel geopotential computation," *AAS/AIAA Space Flight Mechanics Meeting*, vol. AAS 11-158, 2011.

[113] Russell, R. P. and Lam, T., "Designing capture trajectories to unstable periodic orbits around europa," *Journal of the Guidance,Control and Dynamics*, vol. 30, pp. 482–491, 2006.

[114] Russell, R. P. and Lara, M., "Long-life lunar repeat ground track orbits," *Journal of Guidance, Control, and Dynamics*, vol. 30, pp. 982–993, 2007.

[115] RUSSELL, R. P. and OCAMPO, C. A., "Geometric analysis of free-return trajectories following a gravity-assisted flyby," *Journal of Spacecraft and Rockets*, vol. 42, no. 1, pp. 138–151, 2005.

[116] RUSSELL, R. P. and OCAMPO, C. A., "Global search for idealized free-return earth-mars cyclers," *Journal of Guidance, Control, and Dynamics*, vol. 28, no. 2, pp. 194–208, 2005.

[117] SARAMGO, S. F. P. and JR, S. V., "Optimization of the trajectory planning of robot manipulators taking into account the dynamics of the system," *Mechanism and machine theory*, pp. 883–894, 1998.

[118] SAUER, C., "Midas: Mission design and analysis software for the optimization of ballistic interplanetary trajectories," *Journal of the Astronautical Sciences*, vol. 37, pp. 251–259, 1989.

[119] SAYAN, S. and KIRACI, A., "A numerical optimization algorithm for identification of policy options to rehabilitate a publicly managed, pay-as-you-go based pension system," Computing in Economics and Finance 1999 932, Society for Computational Economics, 1999.

[120] SCOTT, J., "High-speed solution of spacecraft trajectory problems using taylor series integration," *Journal of Spacecraft and Rockets*, vol. 47, no. 1, pp. 199–202, 2010.

[121] SEIDELMANN, P. K., ARCHINAL, B. A., A'HEARN, M. F., CONARD, A., CONSOLMAGNO, G. J., HESTROFFER, D., HILTON, J. L., KRASINSKY, G. A., NEUMANN, G., OBERST, J., STOOKE, P., TEDESCO, E. F., THOLEN, D. J., THOMAS, P. C., and WILLIAMS, I., "Report of the iau/iag working group on cartographic coordinates and rotational elements: 2006," *Celestial Mechanics and Dynamical Astronomy*, vol. 98, pp. 155–180, 2007.

[122] SENGUPTA, P., VADALI, S. R., and ALFRIEND, K. T., "Second-order state transition for relative motion near perturbed, elliptic orbits," *Celestial Mechanics and Dynamical Astronomy*, vol. 97, pp. 101–129, 2006.

[123] SHEN, H. and TSIOTRAS, P., "Optimal two-impulse rendezvous between two circular orbits using multiple-revolution lambert's solutions," *Journal of Guidance, Control, and Dynamics*, vol. 26, pp. 50–61, 2003.

[124] SHOEMAKE, K., "Animation rotation with quaternion curves," *Computer Graphics*, vol. 19, pp. 245–254, 1985.

[125] SHUSTER, M., "A survey of attitude representations," *Journal of the Astronautical Sciences*, vol. 41, pp. 439–517, 1993.

[126] SPEDDING, G. R. and RIGNOT, E. J. M., "Performance analysis and application of grid interpolation techniques for fluid flows," *Experiments in Fluids*, vol. 15, pp. 417–430, 1993.

[127] SPIER, G., "Design and implementation of models for the double precision trajectory program (dptraj)," *Technical Memorandum*, vol. 33-451, 1971.

[128] STANDISH, E. M., "JPL planetary and lunar ephemerides," in *CD-ROM, Willman-Bell Inc.*, 1997.

[129] STEWART, G. W., *Afternotes on Numerical Analysis, Lecture 20*, ch. 20, pp. 147–153. 1996.

[130] STRANGE, N. J. and LONGUSKI, J. M., "Graphical method for gravity-assist trajectory design," *Journal of Spacecraft and Rockets*, vol. 39, no. 1, 2002.

[131] SUNDMAN, K. F., "Memoire sue le probleme des trois corps," *Acta Mathmatica*, vol. 36, pp. 105–179, 1912.

[132] TAPLEY, B., RIES, J., BETTADPUR, S., CHAMBERS, D., CHENG, M., CONDI, F., and POOLE, S., "The ggm03 mean earth gravity model from grace," *Eos Trans. AGU, Fall Meet. Suppl.*, vol. 88, no. 52, 2007.

[133] THORNE, J. D., "Lambert's theorem - a complete series solution," *Advances in the Astronautical Sciences*, vol. 119, no. 3, pp. 3061–3074, 2004.

[134] TSIOTRAS, P., SCHAUB, H., and JUNKINS, J. L., "Higher-order cayley transforms with applications to attitude representations," *Journal of the Guidance,Control and Dynamics*, vol. 20, 1997.

[135] UFIMTSEV, I. S. and MARTINEZ, T. J., "Quantum chemistry on graphical processing units. 1. strategies for two-electron integral evaluation," *Journal of Chemical Theory and Computation*, vol. 4, pp. 222–231, 2008.

[136] UMESHA, P., VENURAJU, M., HARTMANN, D., and LEIMBACH, K., "Parallel computing techniques for sensitivity analysis in optimum structural design," *Journal of Computing in Civil Engineering*, vol. 21, no. 6, pp. 463–477, 2007.

[137] UNSER, M., "Splines: a perfect fit for signal and image processing," *IEEE Signal Processing Magazine*, vol. 15, pp. 22–38, 1993.

[138] UPHOFF, C., ROBERTS, P. H., and FRIEDMAN, L. D., "Orbit design concepts for jupiter orbiter missions," *Journal of Spacecraft and Rockets*, vol. 13, no. 6, pp. 348–355, 1976.

[139] VALLADO, D. and MCCLAIN, D. W., *Fundamentals of astrodynamics and applications.* Springer, 2001.

[140] WERNER, R. and SCHEERES, D., "Exterior gravitation of a polyhedron derived and compared with harmonic and mascon gravitation representations of asteroid

4769 castalia," *Journal of Geophysical Research*, vol. 81, no. 26, pp. 4933–4936, 1976.

[141] WHIFFEN, G. J., "Jupiter icy moons orbiter reference trajectory," *AAS/AIAA Astrodynamic Specialist Conference*, vol. 186, 2006.

[142] WHIFFEN, G. J. and SIMS, J. A., "Application of a novel optimal control algorithm to low-thrust trajectory optimization," *Proceeding of the 11th Annual AAS/AIAA Space Flight Mechanics Meeting*, pp. 1524–1540, 2001.

[143] WITTENBRINK, C., KILGARIFF, E., and PRABHU, A., "Fermi gf100 gpu architecture," *Micro, IEEE*, vol. 31, no. 2, pp. 50–59, 2011.

[144] WONG, L., BUECHLER, G., DOWNS, W., SJOGREN, W., and GOTTLIEB, P. M. P., "A surfacelayer representation of the lunar gravitational field," *Journal of Geophysical Research*, vol. 76, no. 26, pp. 6220–6236.

[145] WOODBURN, J., V.SZEBEHELY, and ZARE, K., "An alternative representation of the geopotential," in *AAS/AIAA Spaceflight Mechanics Meeting*, (Albuquerque, NM), pp. 95–191, 1995.

[146] ZHU, W. and PETZOLD, L., "Parallel sensitivity analysis for daes with many parameters," *Concurrency-practice and Experience*, vol. 11, p. 571–585, Aug 1999.

VITA

Nitin Arora was born and spent his childhood in city of Chandigarh, India. He graduated from "Punjab Engineering College" with a Bachelors degree in Civil Engineering in 2007. In fall 2007, Nitin got the opportunity to pursue his studies in Aerospace Engineering at the Georgia Institute of Technology under the guidance of Dr. Ryan P. Russell. He received a Master's Degree in Aerospace Engineering from Georgia Institute of Technology in 2009 and then began working towards his PhD. Nitin was also a visiting research scholar at Center for Space Research at The University of Texas at Austin and an intern at NASA Jet Propulsion Lab, in 2012. Over the course of his PhD. Nitin has been a member of the Space Systems Design Lab at the Georgia Institute of Technology.