Standardization of Medical Imaging data from non-uniform sources within the Julia language ecosystem

Divyansh Goyal*[1], Hariansh Vashisht*[2]
Department of Artificial Intelligence and Machine Learning
Guru Gobind Singh Indraprastha University
New Delhi, India
{divital2004, harianshvashisht24155}@gmail.com

Dr. Suman Bhatia
Department of Artificial Intelligence and Machine Learning
Dr. Akhilesh Das Gupta Institute of Professional Studies
Shahdara, New Delhi, India
suman.bhatia@adgitmdelhi.ac.in

Abstract

Medical imaging has become critical for diagnosis, treatment planning and follow-up across modalities like CT, MRI, PET, but lack of standardization poses barriers for computational analysis including adoption of cutting-edge AI techniques like deep learning and radiomics. The standardization of medical imaging data from non-uniform sources is a critical issue due to the large number of medical imaging data formats, posing challenges for reproducibility and interoperability. This research paper addresses the challenges posed by non-uniform data formats, data inhomogeneity, and complex preprocessing in the context of medical imaging. The paper aims to propose and evaluate methods for standardizing medical imaging data to improve reproducibility and facilitate cross-platform and cross-institutional data analysis. The research addresses the need for retrospective harmonization of heterogeneous data and the utilization of standards such as the Neuroimaging Informatics Technology Initiative (NIFTI) to ensure interoperability. Julia combines high-level dynamic programmability with ability to compile to efficient machine code and parallel execution, offering order of magnitude acceleration versus traditional platforms for technical computing like Python and Matlab.

***Index Terms*— Keywords: Generative AI, Medical Imaging, NIFTI, Image Segmentation, Julia**

I. INTRODUCTION

Medical imaging has revolutionized healthcare by enabling non-invasive visualization of anatomical structures and biological processes for diagnostic and therapeutic purposes. Various modalities like X-ray, CT, MRI, and PET are routinely used in clinical workflows generating vast amounts of complex, multi-dimensional data [1]. This data needs standardized representation across systems to extract clinically useful biomarkers through computational analysis including emerging AI techniques. However, interoperability remains challenging due to proprietary formats closely tied to scanner hardware, optimized for acquisition rather than analysis.

The Digital Imaging and Communications in Medicine (DICOM) standard specifies file formats, storage media exchange protocols to promote integration of imaging equipment like scanners and printers from multiple manufacturers (Mildenberger et al., 2002). But DICOM has limitations for computational analysis with recent extensions not backward compatible and still tied to specific hardware [**?**]. Conversion to more analysis-friendly formats like NIfTI (Neuroimaging Informatics Technology Initiative) and

NRRD (Nearly Raw Raster Data) solve these issues but have tradeoffs losing metadata and risk distortions. Building standardized end-to-end imaging pipelines remains an open challenge requiring extensive research as discussed in subsequent sections.

This paper focuses on the potential of an emerging heterogenous technical computing language known as Julia that offers specific advantages for medical imaging versus commonly used platforms like Python or Matlab. Julia enables easy interoperability with existing code while providing order of magnitude performance gains through just-in-time compilation, distributed parallel execution and focus on technical computing (Bezanson et al., 2017) [**?**]. We critically evaluate ongoing Julia research for tasks spanning standardized data representation, AI model development and radiomic analysis to identify open challenges.

This paper is structured as follows: Section II provides details on the literature survey. Section III explores the problems as a result of non-uniform nature of data *heterogenous* problem. In Section IV the details about experimental results have been provided; followed by conclusions and references. Appendix provides the performance statistics of the Barnes-Hut algorithm in OpenMP on galactic datasets chosen from Princeton University [**?**]; with number of bodies ranging from 5 to 30,002.

II. LITERATURE SURVEY

A. Background

Imaging data exists in a myriad of proprietary formats designed for acquisition and storage rather than interoperability and analysis. The Digital Imaging and Communications in Medicine (DICOM) standard by National Electrical Manufacturers Association (NEMA) defines storage formats as well service protocols for communication between imaging devices and archives (Mildenberger et al., 2002). However limitations still exist when attempting to extract imaging data for further computational analysis. Data needs to be converted into alternate structures like the Nearly Raw Raster Data (NRRD) format or the Neuroimaging Informatics Technology Initiative (NIFTI) format. Such conversions can be complex, time consuming and often result in loss of metadata or distortions.

The DICOM standard has undoubtedly provided some level of interoperability for medical imaging but has significant limitations for emerging use cases in quantitative image analysis and AI as per recent research. Ferrante et al. (2018) proposed DICODeep, an extension of DICOM format and network protocol to add support for deep learning workflows. Custom extensions are suggested for annotations, hierarchical labels, inference reports and logging dataset provenance. But these would not be backward compatible with existing DICOM archives requiring additional translation modules. DICOM is also tightly bound to specific modalities and proprietary scanner hardware which hampers adoption.

B. Literature Survey

Medical imaging data is broadly comprised of two interrelated components - the actual pixel or voxel values representing anatomical structures and metadata describing acquisition context, techncial parameters and other ancillary information essential for proper interpretation (Smith and Smith, 2021).

Standardized frameworks like DICOM have long focused on metadata header specifications enabling some level of multi-vendor interoperability on imaging devices (Mildenberger et al., 2002). However limitations exist when attempting to reliably extract such data for downstream computational tasks, raising barriers for techniques like radiomics analysis (Kumar et al., 2015). DICOM tags may not fully capture relevant clinical or demographic attributes requiring error prone mapping from related reports. Varying terminology for sequences and protocols also introduce heterogeneity not conducive for algorithms expecting homogenous inputs.

DICOM, *ANALYZE 7.5*, *NIFTI*, *NRRD*, *HDF5* and *MINC* are examples of some of the most prominent Medical Imaging file formats out there.

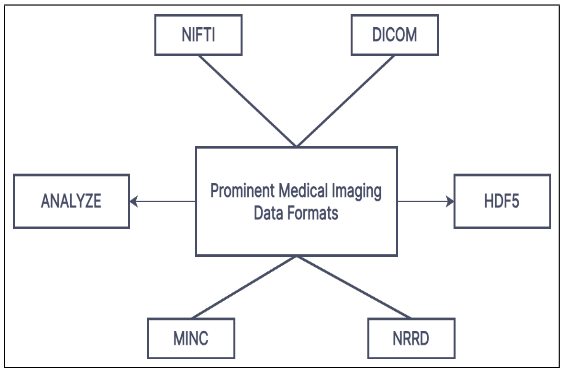


Fig. 1. Medical Imaging Data Formats *NIfTI (Neuroimaging Informatics Technology Initiative), DICOM (Digital Imaging and Communications in Medicine), Analyze, HDF5 (Hierarchical Data Format version 5), MINC (Medical Imaging NetCDF), NRRD (Nearly Raw Raster Data)*

The Neuroimaging Informatics Technology Initiative (NIfTI) format was developed in the early 2000s by the National Institutes of Health (NIH) to overcome limitations of the previously prevalent Analyze 7.5 standard for medical imaging data storage and analysis. A key drawback of Analyze 7.5 identified was the use of detached header and data files ("img" and "hdr") which increased complexity for transferring datasets between platforms (Keator et al., 2016).

To promote interoperability, NIfTI allows consolidating metadata and voxel intensity values into a single ".nii" file with the option to compress during network transfers. The unified structure simplifies workflow automation and avoids potential errors of header-data mismatch in Analyze formats. Backward compatibility is retained by keeping header size and parameters identical to Analyze 7.5 as originally reported in literature (Li et al., 2016).

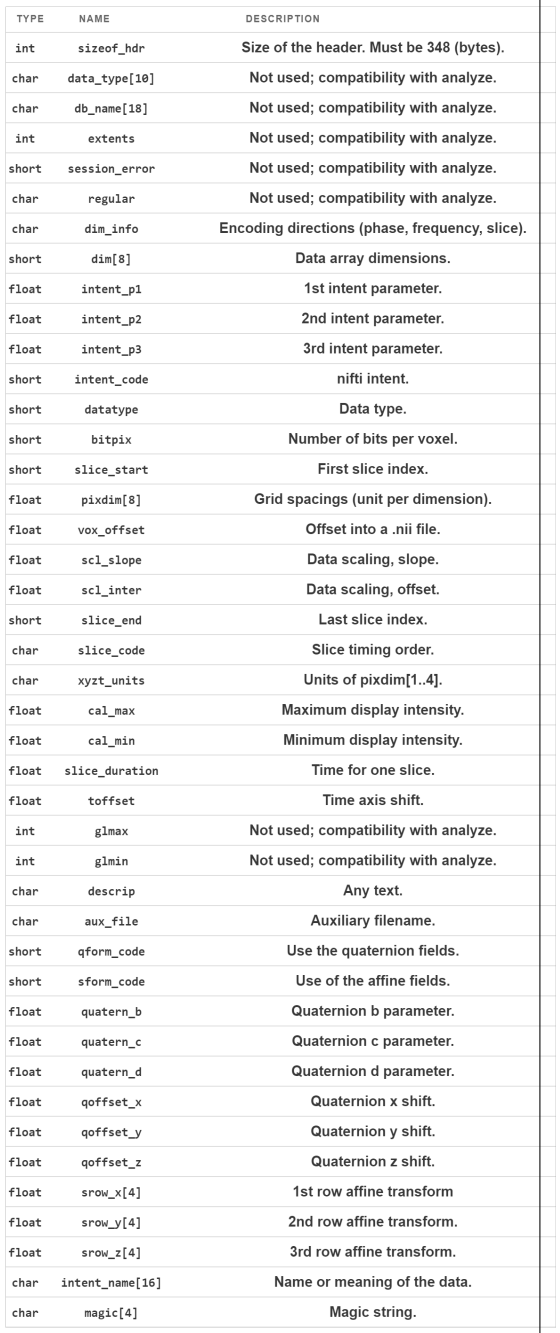
A notable enhancement is utilizing two affine matrices within the NIfTI header to accurately relate voxel indices to spatial coordinates across acquisition scanners and mappings between anatomical and functional images (Van Essen et al., 2001). This allows retaining provenance across processing stages which may involve non-linear transformations.

While originally designed for brain imaging, NIfTI offers advantages for standardized representation of multidimensional imaging data across domains through its integrated structure. Recent work by Pan et al. (2021) demonstrates adoption for cardiac and abdomen MRI segmentation workflows with deep learning and analytics seamlessly integrating NIfTI volumes reconstructed from vendor DICOM formats.

NIfTI format aims to promote standardization for computational analysis tasks by unifying voxel data with essential metadata in an interoperable structure preserving spatial and semantic context across platforms. This shows its potential as a standardized intermediate representation even when upstream systems have modality or proprietary encodings.

C. Working with NIfti files with the help of Nibabel and ITK

Efficient analysis of volumetric NIfTI files involves seamless interoperability with computational pipelines reading, manipulating and visualizing imaging data. NiBabel, an open-source Python package, has emerged as a prominent utility that handles interfacing between diverse file formats and in-memory analytic objects like NumPy arrays (Brett et al., 2020a). It insulates data science code from underlying storage specifics allowing researchers to focus on algorithm implementation. The simplicity of nib.load() and nib.save() calls facilitates rapid prototyping and experimentation.



TYPE	NAME	DESCRIPTION
int	sizeof_hdr	Size of the header. Must be 348 (bytes).
char	data_type[10]	Not used; compatibility with analyze.
char	db_name[18]	Not used; compatibility with analyze.
int	extents	Not used; compatibility with analyze.
short	session_error	Not used; compatibility with analyze.
char	regular	Not used; compatibility with analyze.
char	dim_info	Encoding directions (phase, frequency, slice).
short	dim[8]	Data array dimensions.
float	intent_p1	1st intent parameter.
float	intent_p2	2nd intent parameter.
float	intent_p3	3rd intent parameter.
short	intent_code	nifti intent.
short	datatype	Data type.
short	bitpix	Number of bits per voxel.
short	slice_start	First slice index.
float	pixdim[8]	Grid spacings (unit per dimension).
float	vox_offset	Offset into a .nii file.
float	scl_slope	Data scaling, slope.
float	scl_inter	Data scaling, offset.
short	slice_end	Last slice index.
char	slice_code	Slice timing order.
char	xyzt_units	Units of pixdim[1..4].
float	cal_max	Maximum display intensity.
float	cal_min	Minimum display intensity.
float	slice_duration	Time for one slice.
float	toffset	Time axis shift.
int	glmax	Not used; compatibility with analyze.
int	glmin	Not used; compatibility with analyze.
char	descrip	Any text.
char	aux_file	Auxiliary filename.
short	qform_code	Use the quaternion fields.
short	sform_code	Use of the affine fields.
float	quatern_b	Quaternion b parameter.
float	quatern_c	Quaternion c parameter.
float	quatern_d	Quaternion d parameter.
float	qoffset_x	Quaternion x shift.
float	qoffset_y	Quaternion y shift.
float	qoffset_z	Quaternion z shift.
float	srow_x[4]	1st row affine transform
float	srow_y[4]	2nd row affine transform.
float	srow_z[4]	3rd row affine transform.
char	intent_name[16]	Name or meaning of the data.
char	magic[4]	Magic string.

Fig. 2. A NIfTI file Header Structure

NiBabel development is guided by true community stewardship with contributors across academia and industry (Brett et al., 2020b). Support for all modern NIfTI variants ensures consistency and reliability. Another project of similar ethos is the Insight Toolkit (ITK) spearheaded by the National Library of Medicine (McCormick, 2021). ITK provides a templated C++ framework with state-of-the-art algorithms for segmentation, registration and filtering. Interfacing ITK with high productivity languages is crucial for productivity which NiBabel enables without alienating ITK's computational performance. These symbiotic partnerships between academic and commercial stakeholders showcase the importance of unified representations like NIfTI towards advancing medical imaging capabilities through a thriving ecosystem of interoperable modular toolkits.

D. Medical Imaging Data Augmentation

Data augmentation has become a vital technique for training robust deep learning models for medical image analysis. By generating plausible synthetic variations of real images, data augmentation expands limited training sets to improve model generalization. Standard approaches like cropping, flipping, and intensity shifts are prevalent, but recent research has explored more advanced options tailor-made for medical data needs. Strategies leveraging generative adversarial networks (GANs) and interpolation between real scans show promise to reflect complex real-world variations. Augmentation overall appears set to expand as a core pillar enabling practical medical deep learning developments in line with modern best practices.

Medical imaging analysis must contend with limited availability of annotated data which hinders developing robust algorithms. Findings reveal affine transformations are most widely adopted as a result of straightforward implementation. Generative adversarial networks (GANs) and pixel-level distortions also see notable uptake stemming from their ability to simulate images closer to true acquisition process variations.

Based on original data transformation, the following augmentation methodologies exist:

- Affine Transformation
- Erasing Transformation
- Elastic Transformation
- Pixel-Level Transformation

In medical imaging, **affine augmentation** creates realistic variations mimicking differences in patient positioning, anatomical variance, image acquisition, and processing. This introduces robustness without altering underlying pathology. However, constraints are required to avoid non-anatomical shearing and extreme scaling that distort diagnostic features.

Affine transformations are defined by point-wise linear mapping matrices that encode translation, rotation, scaling, shearing, and reflection operations while preserving collinearity and ratios of distances on images. They provide efficient re-sampling and interpolation of pixel intensities without introducing nonlinear distortions.

Key affine transformation matrices include:

- The translation matrix T(tx, ty) correctly encodes shifting an image by tx pixels along x-axis and ty pixels along y-axis.
- The rotation matrix R(θ) with angle θ parameter is the standard 2D rotation matrix formula.
- The scale matrix S(sx, sy) properly scales the x and y dimensions by independent factors sx and sy.
- The shear matrix H(shx, shy) applies the standard formulas for x-shear proportional to y and y-shear proportional to x.

Erasing transformations selectively occlude or alter regions of an image to encourage a model to learn more robust representations less dependent on specific visual features. This technique randomly selects a rectangular or circular subset of pixels and replaces intensity values with grayscale noise, constant values, or random colors. For medical images, selective erasing can reflect naturally occurring variability from artifacts, contrasts, lesions or instrumentation. It also simulates missing regions from limited scanning windows and forces dependency on contextual cues. However, care must be taken to avoid excessive removal of clinically valuable features. Constraints on erase size, location, and background values are necessary to maintain anatomical realism.

Elastic distortions apply localized, nonlinear warping to simulate anatomical deviations and acquisition artifacts. Unlike affine transforms, elasticity preserves topology but not distance or orientation. This reflects real-world variation from tissue deformation, patient motion, imaging distortion, and processing artifacts. The core technique involves constructing a sparse deformation vector field and interpolating pixel intensities based on the distorted coordinate mapping. Constraints are necessary to limit spatial warping and avoid non-anatomical shapes. Careful tuning of random displacement parameters and smoothing kernels is required to mimic naturally observed variation.

Pixel-level transformations directly alter pixel intensities to induce variability reflecting acquisition pa-

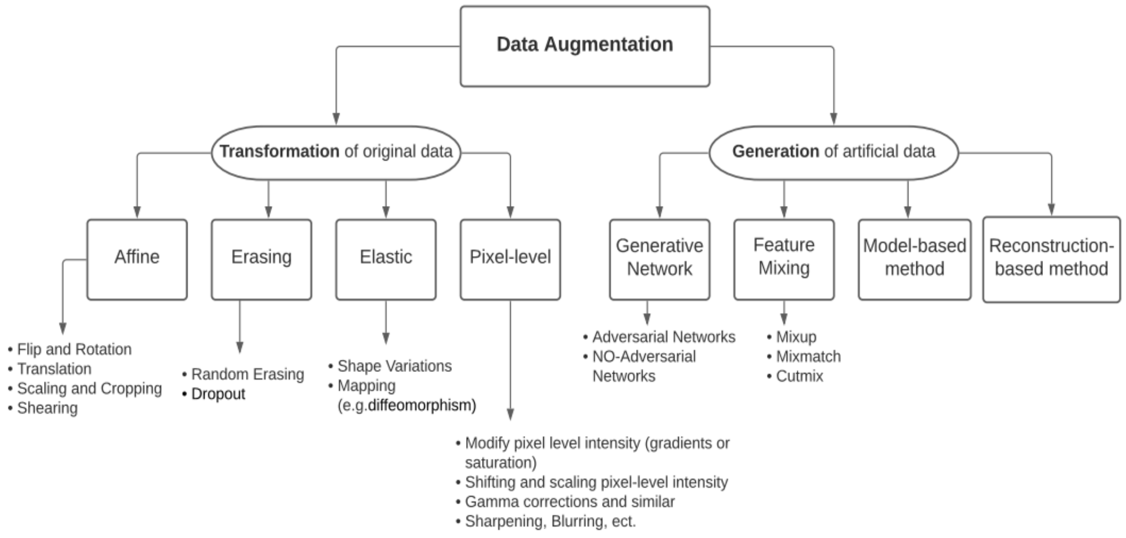

Data Augmentation
Transformation of original data
Generation of artificial data
Affine
Erasing
Elastic
Pixel-level
Generative Network
Feature Mixing
Model-based method
Reconstruction-based method
• Flip and Rotation
• Translation
• Scaling and Cropping
• Shearing
• Random Erasing
• Dropout
• Shape Variations
• Mapping (e.g.diffeomorphism)
• Modify pixel level intensity (gradients or saturation)
• Shifting and scaling pixel-level intensity
• Gamma corrections and similar
• Sharpening, Blurring, ect.
• Adversarial Networks
• NO-Adversarial Networks
• Mixup
• Mixmatch
• Cutmix

Fig. 3. Data Augmentation Techniques

rameters, scanner settings, and physiological factors. Common approaches include:

- Brightness shifts: Adding or clipping intensity ranges. Can mimic underexposure or quantification variability between devices. Constraints prevent wrapping out-of-bounds.
- Contrast enhancement: Rescaling intensity gradients using sigmoid/tanh-based mappings. Accounts for variability in tissue visual differentiation.
- Blurring: Smoothing with Gaussian or other convolutional filters to simulate motion or depth of field. Kernel sizes and weight distributions tune effect strength.
- Noise injection: Adding correlated noise patterns through procedural (Perlin) functions or uncorrelated Gaussian/Poisson distributions. Introduces acquisition artifacts at controllable levels.
- Spatial transforms: Applying affine/elastic warps before interpolation to combine geometric and pixel distortions. Enables efficient coordinated augmentation.

One thing to be noted is that, with all augmentation strategies, validating final data integrity is crucial for medical settings.

E. Parallelization of All-Pairs Algorithm (OpenMP)

The brute force algorithm is very easily parallelized as it is known in advance exactly how much work needs to be done. The work can be partitioned easily among processes with a block partitioning strategy. The number of bodies is known and to update each body takes the same amount of calculation. Assigning each process to calculate a block of bodies each $\frac{number of planets}{number of processors}$ in size will mean each process performs the same amount of calculations. Therefore partitioning the workload effectively; depicted by Algorithm 2.

F. Parallelization of All-Pairs Algorithm (CUDA)

Algorithm 3 given below is the parallel All-Pairs algorithm used to compute forces on all bodies using Nvidia CUDA.

G. Sequential Barnes-Hut Algorithm

The Barnes-Hut algorithm [**?**] approximates a solution to the gravitational N-Body problem by clustering groups of distant bodies together as a single pseudo-body. Each has an overall mass and center of mass based on the individual bodies it contains. It achieves this by creating a tree structure where each node has four children and each node has a center of mass and total mass based on that of its children. When created, this tree describes the whole system where each internal node represents a pseudo-body. Each star then uses the tree to work out the forces it experiences. The algorithm to realize the spatial system into a tree structure is achieved as follows:

- Divide the whole domain into four square regions of equal size.

- If any of these regions contains more than one body, recursively divide that region into four more squares. Continue until each square contains maximum of one body.
- Once the tree is created perform a recursive walk to calculate the center of mass, $\vec{c}$, as in (**??**), where m_i is the mass of a node´s i^{th} child and $\vec{c}_i$ is the center of mass of a node´s i^{th} child. .

This creates a tree where the root node (Fig. 1) contains the whole system. Each node has four children (Fig. 2) and the leaves of the tree are the individual bodies [**?**]. The construction of the tree can be done with $O(N\log(N))$ runtime. Each body now uses this tree to calculate the acceleration it experiences due to every other body. The force calculation is performed for each body. It recursively finds nodes in the tree which are considered to be far enough away to perform an interaction with. The calculation to decide whether a node is far enough away is called the *opening condition.* It is important because it decides how many bodies can be grouped together as a pseudo-body; the more bodies which are grouped together, the less accurate the calculations will be. The opening condition is a simple relationship, $\frac{l}{D} < \theta$, where θ is the fixed accuracy parameter which is positive, l is the width of the current internal node and D is the distance of the body from the center of of mass of the current node to the body the force is being calculated for. The sequential algorithm for the same is given as Algorithm 4.

H. Parallel Barnes-Hut Algorithm using OpenMP– Force Computation is Parallelized (Method-1)

- Build the quad-tree.
- Calculate the center of mass for all cells.
- Traverse the quad-tree.
- Calculate the force on the nodes.

I. Parallel Barnes-Hut Algorithm using OpenMP– Mass Distribution is Parallelized (Method-2)

III. WORK DONE AND RESULTS ANALYSIS

This section focuses on running each algorithm in serial and in parallel. Testing Speedup, Cost and Efficiency (see equations (1), (2), (3)). All the tests for All-Pairs algorithm (openMP) are ran on nearly identical machines with the following specification:

- Model– Asus
- Processor– i5 7200U @ 4x 3.1GHz
- Memory– 8GB DDR3 at 1333MHz
- Network– 10/100/1000 Gigabit LAN Connection
- Operating System– Arch Linux

All the tests for All-Pairs algorithm (CUDA) are ran on nearly identical machines with the following specification:

- Nvidia Tesla Server
- Operating System– Ubuntu Linux

All the tests for Barnes-Hut algorithm (openMP, both methods) are ran on nearly identical machines with the following specification:

- Model– HP
- Processor– i5 6200U
- Memory– 8GB DDR3 @ 2.8GHz
- Network– 10/100/1000 Gigabit LAN Connection
- Operating System– Ubuntu Linux

$$Speedup(S) = \frac{TimeforSerialExecution}{TimeforParallelExecution} \quad (1)$$

$$Cost(C) = ParallelExecution \times NumberofProcessors \quad (2)$$

$$Efficiency(E) = \frac{Speedup}{TimeforParallelExecution} \quad (3)$$

A. Result and Analysis of Parallel All-Pairs algorithm in OpenMP and Nvidia CUDA

Images/fig_3.png

Fig. 4. Serial (black) vs. Parallel (red) execution for *Planets.txt* [**?**] for 5 bodies using OpenMP (Algorithm 2)

B. Result and Analysis of Parallel Barnes-Hut Algorithm using OpenMP

In Fig.9, the sequential algorithm proved to be efficient as compared to OpenMP. One explanation for the same would include the communication overhead and thread overhead for such small dataset (5 bodies). In Fig. 10, it was observed that the OpenMP implementation performed better in the case of 2 and 4 threads but the overheads increased as the number of threads increased from 4. In the case of Fig. 11, the Barnes-Hut implementation in OpenMP algorithm performed better in the case of 2 threads but increased progressively before reducing once in 8 threads and increasing again.

IV. CONCLUSIONS

In this paper, we analyzed two algorithms to solve the classical N-Body problem– the naive All-Pairs Algorithm and quad-tree based Barnes-Hut Algorithm in OpenMP and CUDA. Compared to the sequential execution we noticed a decrease in execution time till a certain level of parallelization, after which the time either remained the same or increased. The performance of these algorithms can be further bettered by running the algorithms on a processor with a higher multiprogramming support.

The parallel direct method scaled linearly with respect to the number of processes. The communication overhead for the parallel direct method is negligible as the number of stars is so small, but as more processes are added the algorithm becomes plausible for larger numbers of N, but with increasing N will come increasing communication overheads. Due to limitations with time this project only implemented a simple parallel version of the Barnes-Hut algorithm that contains no load balancing. The computation time of the Parallel Barnes-Hut scaled almost linearly, but with an increasing number of processes came an increasing communication overhead which soon outweighed the benefit seen due to the increase in computation time. The Barnes-Hut algorithm can offer substantial increases in running time depending on the choice of θ. This shows how well the naive method is improved by parallel computing.

APPENDIX

The appendix shows the analysis of the Barnes-Hut algorithm implemented in Parallel using OpenMP (method-1). The Sequential and Parallel times have been shown in Table 1; for all the galactic datasets [**?**] with number of bodies ranging from 5 to 30002.

ACKNOWLEDGMENT

We would like to thank Dr. Geetha V for her valuable comments and suggestions to improve the quality of the paper. We are also grateful to Miss Archana for helping us review our work regularly. We would also like to thank the Department of Information Technology, NITK Surathkal for providing us with Tesla GPU for us to test our code.

REFERENCES

[1] Wikipedia. (2018) N-body problem. [Accessed 7 Jan. 2018]. [Online]. Available: https://en.wikipedia.org/wiki/N-body_problem

TABLE I

PERFORMANCE OF SERIAL CODE VS. PARALLEL CODE ON GALACTIC DATASETS [?] OF BARNES-HUT ALGORITHM IN OPENMP (METHOD-1)

Dataset	**Number of Particles**	**Serial Time (seconds)**	**Parallel Time (seconds)**					
			Number of Threads					
			1	**2**	**4**	**8**	**16**	**32**
asteroids1000.txt	1000	0.023097	0.020348	0.021905	0.030464	0.063325	0.121116	0.221256
cluster2582.txt	2582	0.004927	0.005837	0.005042	0.011231	0.008328	0.011733	0.014243
collision1.txt	2000	0.004917	0.004829	0.004447	0.006030	0.005751	0.009468	0.012608
collision2.txt	2002	0.006227	0.006008	0.006098	0.006309	0.006821	0.009951	0.013182
galaxy1.txt	802	0.015414	0.015616	0.015217	0.020928	0.045315	0.072090	0.110689
galaxy2.txt	652	0.012274	0.012615	0.014664	0.023931	0.028826	0.040485	0.072064
galaxy3.txt	2001	0.091639	0.087738	0.094466	0.141264	0.264529	0.488200	0.975077
galaxy4.txt	502	0.012875	0.013325	0.010431	0.012065	0.027304	0.037397	0.051786
galaxy10k.txt	10001	2.325312	2.357691	2.422882	3.557520	6.697054	13.312913	27.061886
galaxy20k.txt	20001	13.663441	15.492622	16.259973	23.813991	45.588013	88.301931	160.741782
galaxy30k.txt	30002	0.032405	0.032411	0.031647	0.057545	0.050811	0.075314	0.171779
galaxyform2500.txt	2500	0.007052	0.005922	0.006162	0.006707	0.008641	0.011501	0.016563
galaxymerge1.txt	2000	0.004920	0.005160	0.004812	0.006784	0.006742	0.008701	0.018789
galaxymerge2.txt	4000	0.011205	0.010364	0.003930	0.012193	0.011976	0.018860	0.024891
galaxymerge3.txt	2901	0.009433	0.009095	0.009045	0.015460	0.011692	0.012852	0.019202
planets.txt	5	0.000070	0.000120	0.000406	0.001250	0.001246	0.001997	0.003918
saturnrings.txt	11987	0.024471	0.024749	0.020095	0.025863	0.032043	0.038763	0.064468
spiralgalaxy.txt	843	0.017879	0.017627	0.023605	0.024740	0.052584	0.091534	0.166260