Practical Pedagogical Approaches in HPC for Optimizing Complex Programming Algorithms: A Nuclear Fusion Application Case Study

Someswar Dutta
SST-1 Operation Division
Institute for Plasma Research (IPR)
Gandhinagar, India
sdutta@ipr.res.in

Deepak Aggarwal
Computer Division
Institute for Plasma Research (IPR)
Gandhinagar, India
devang.twister@gmail.com

Daniel Raju
SST-1 Operation Division
IPR, Gandhinagar, India
HBNI, Mumbai, India
raju@ipr.res.in

***Abstract*—High-performance computing (HPC) systems are enabling scientists and researchers to address diverse challenges, from sustainable and renewable energy sources and astrophysics to weather prediction, material innovation and many more. Therefore it is crucial to develop appropriate pedagogy and train the young minds to exploit the ever expanding computing power of the HPC systems in solving the critical problems. This article describes a practical hands-on approach to enhance and optimise the performance of a nuclear fusion energy application code. The article discusses the importance of the necessity of refactorization and profiling of the application, choice of compiler to the merit of selecting the embarrassingly parallel computation topology to harness the computational power of the HPC and enhance the performance of the application. The study reveals that code restructuring and improved I/O communication between subroutine calls, coupled with the adoption of a modern compiler (Intel IFORT) instead of the legacy compiler (Lahey/Fujitsu Fortran 95 2004), notably bolster the application's performance. In addition to the performance enhancements, this exercise also enhanced the code's portability, making it compatible with any HPC system equipped with modern compilers. Moreover, deploying the optimised serial application via job scheduler facilitated by *Array Job* that leverages the embarrassingly parallel computation topology on the Institute's HPC facility, yields a substantial performance boost, surpassing the base version of the application by orders of magnitude. The methodology described here can easily be adopted by the early researchers and code developers to enhance the performance of any such code designed for problems characterised by the discussed algorithmic topology.**

***Index Terms*—Performance optimization, Parallel computations, High Performance Computing, HPC Pedagogy, PDC**

I. Introduction

High-performance computing (HPC) systems have emerged as indispensable tools for scientists and researchers, facilitating solutions to multifaceted challenges [1]. These encompass areas such as sustainable and renewable energy sources [2], astrophysical exploration [3], weather forecasting [4], material advancement [5], fusion plasma physics [6], [7] and beyond [8]. Given their broad impact, there's an increasing need to cultivate adept pedagogical methods to impart HPC education and empower emerging talents. This involves equipping young minds with the skills to harness the escalating computational potential of HPC systems. Inclusion of real-world examples in teaching would illustrate the practical applications of HPC. Emphasis on parallel and distributed computing (PDC) and performance optimization is crucial, alongside understanding cluster architecture and the use of HPC tools. This multifaceted approach would equip young minds to harness HPC's computational power and solve complex challenges [9], [10]. Code optimization is a diverse area of prime importance in HPC research and is an integral part of any code development. This involves code refactorization and rearrangements, memory I/O optimization, selection of right compiler with appropriate compiler options and selection of suitable parallel and distributed programming topology [11–14], to enhance the performance of the code. While theoretical understanding imparts knowledge about HPC topics, a practical hands-on approach equips students with the ability to apply this knowledge to real-world problems, providing them with a valuable edge. In this article pedagogical approaches are discussed on how to overcome the challenges in teaching these abstract concepts of HPC pedagogy with practical hands-on approaches. To do so how step by step processes followed to optimize a real world fusion physics application code for HPC system has been discussed.

The fusion application code discussed here deals with the high energy runaway electron (RE) dynamics in nuclear fusion machines namely tokamak machines [15], which is an extremely important phenomenon as it poses severe threats to damage the machine components thereby reducing its operational lifetime. We have developed a code named PARTICLE-3D (P3D) [16] to simulate RE dynamics in nuclear fusion machines. The initially developed serial base version of the code was time-consuming for simulations involving a large number of electrons. Through optimization and parallelization techniques, the code's performance is significantly improved, making it suitable for advanced tokamak research. The optimised P3D code is now valuable for conducting high-energy particle dynamics analysis and control studies in various tokamak machines, addressing critical issues in nuclear fusion research [17]. The methodology followed in this

article to optimise and parallelize the code would be adopted to train the young researchers, Ph.D students, scientists and college students in the field of HPC and PDC concepts as part of IPR's HPC outreach program [18]. The rest of the article is organised as follows. In section II, previously carried out related works and scope of the present work are discussed. Section III delves into the challenges pertaining to HPC pedagogy that are addressed in this article, Section IV includes the introduction of the P3D code along with its algorithmic topology and workflow. Section V details hands-on approaches to optimise the performance of the P3D code and the performance comparison results are detailed in Section VI. Finally, summary and conclusions are drawn in section VII.

II. Related work and scope

Several works had been carried out to design, build and teach HPC/PDC through different approaches and methods. In N. Wang et al. [19], authors discuss the necessity of transitioning to parallel computing due to the limitations of serial computation in the field of mechanical engineering. The study demonstrates the effectiveness of parallelization based on embarrassingly parallel topology, even on small clusters, for substantial speedup in execution time. Another study [20] discusses "HPC/PDC Immunization" at the introductory computer science level, using threaded solutions to introduce parallel programming, in which students experience CPU-bound problems, leading to a desire for further parallel programming education. In the field of computational biology applications, highlighting HPC's value in biological problem-solving, a course had been designed as discussed in V. Sravanan et al. [21]. In Institute for Plasma Research, Gandhinagar, India, efforts have been made to address teaching parallel and distributed programming, with a focus on clarifying fundamental concepts for students , emphasising the ongoing need for effective pedagogy [22]. Another work comprising an elaborate review on the feasibility study on developing parallel applications using small-scale clusters of single-board computers had been carried out in Deepak et al. [18]. The approach discussed in that article aids in learning cluster architecture and parallel programming and scalability studies, which are incorporated in the HPC outreach program to train the young minds.

The present work discusses a hands-on approach to enhance performance of a nuclear fusion physics application code through optimization and parallelization, in view of building an effective HPC/PDC learning by putting together several key elements of complex code development. It highlights the necessity of code refactorization and profiling, the importance of compiler selection to enhance code portability (e.g., Intel Ifort over legacy compilers), and the benefits of choosing embarrassingly parallel (depending on the algorithm) computation implemented by *Array Job* functionality of the job scheduler over the OMP and MPI implementation. While previous works have focused on specific aspects or techniques, this study provides a more unified perspective, making it an invaluable resource for early researchers and developers in the field of nuclear fusion research. Finally, integrating the overall methodology discussed here with the IPR's HPC outreach program would offer valuable insights for early researchers and developers seeking to develop, benchmark, optimise and parallelize code for algorithmically similar problems in the HPC/PDC domain.

III. HPC Pedagogy: Challenges addressed

High-Performance Computing (HPC) is a vast field encompassing parallel and distributed computing, optimization techniques, playing a crucial role in solving complex problems across various domains. However, teaching HPC concepts can be challenging. Instructors often face hurdles in making such complex, abstract topics easily presentable, understandable and accessible to the students. Therefore, suitable HPC pedagogy needs to be adopted that has a comprehensive approach to equip students with the HPC related knowledge and practical skills needed to excel in the world of high-performance computing. In the present work it is proposed that, such challenges in teaching the HPC concepts can be addressed by incorporating following strategies:

- **Hands-on training:** Theory alone is insufficient. Practical, hands-on training is essential for students to grasp these concepts. Instructors provided opportunities for students to apply their knowledge in real-world scenarios, fostering a deeper understanding.
- **Interactive learning:** Engaging students through interactive learning methods, like coding exercises and simulations, keeps them motivated and helps demystify complex topics.
- **Real-world examples:** Using real-world examples and case studies demonstrates the practical relevance of these concepts. Showing how code portability, compiler selection, profilers, and optimization are used in actual HPC projects can inspire students.

The present work delves into emphasizing how the above mentioned strategies can be applied to teach some specific critical aspects of the HPC concepts, as discussed below:

- **Code portability:** In HPC, fostering code portability is vital. Students learn to craft adaptable code for diverse HPC systems, emphasising best practices and standard libraries. Portability ensures application longevity and adaptability of the code for diverse HPC systems. For example: in the present case study, the application initially compiled only with the legacy *LF95* compiler on the old HPC system. Transitioning to the modern *Intel IFORT* compiler on ANTYA required significant changes in syntax and formatting for compatibility.
- **Selection of compiler:** Compiler choice significantly influences code performance; students learn to select the best compiler, optimizing for specific systems. In the present case study, opting for a modern compiler over legacy compiler enhances code portability by making the application compiler agnostic, which allows easy

migration across HPC systems. Also choice of modern compilers along with compatible hardware significantly boost application performance.Students can learn about code portability and the significance of compiler choice by modifying and executing various versions of the application during interactive hands-on sessions.

- **Role of code profilers:** Profilers are indispensable tools for code analysis. Students gain proficiency in using code profilers to identify bottlenecks and inefficiencies within their applications which improves overall code performance. Profiling is a key skill for HPC code developers. The concept and importance of profiling can be taught by actually profiling the different versions of a code to compare the performance of the code versions through hot spot analysis, as carried out and presented later in the article.
- **Code optimization:** Code optimization is the cornerstone of HPC pedagogy. Students learn various techniques to enhance code performance, including parallelization, vectorization, and algorithmic improvements. They acquire the expertise to make their code efficient, which is essential in HPC, where speed is crucial. Through the present case study, students can learn how the hot spot analysis is carried out using code profiler to identify the most time consuming function / algorithm of the application and then subsequently how the modification of that algorithm can be done to enhance the performance of the optimized application.
- **Parallel and distributed computing:** Students should explore parallel algorithms and data structures tailored for HPC environments. This involves understanding strategies for load balancing, data distribution, and communication patterns. Along with that student must develop knowledge to choose the best suited parallelization topology (Embrassingly parallel (EP), OMP and MPI) for their application. During the hands-on session, students learn to develop job scripts for EP computation via the HPC job scheduler's array job deployment facility to harness distributed computing power.

Therefore combining hands-on practice, interactive learning methods, and real-world examples, with comprehensive theoretical education, instructors can address the discussed challenges in HPC pedagogy. Also by doing so, instructors can empower students with the knowledge and skills needed to excel in the dynamic and demanding field of high-performance computing. How the discussed practical pedagogical approaches in understanding the HPC concepts applied to optimize a real world fusion application code for HPC system has been presented in the rest of this article.

IV. Full orbit following code PARTICLE-3D (P3D) : A nuclear fusion energy application

A. *P3D code: what it does?*

Runaway electrons, often referred to as REs, are highly energetic (several MeV) electrons that arise in a tokamak irrespective of their size, across various plasma operation scenarios, that can exert a substantial impact on plasma operation and have the potential to inflict severe damage on both the first wall and in-vessel components of tokamak machines, making the investigation of their dynamics a vital facet of nuclear fusion energy research [23]. As mitigation and control of such REs are extreamly important, to date several codes are written to study the RE dynamics in a tokamak with different perspectives and objectives. To study RE dynamics of an ensemble of up to 6000 REs, these codes often use MPI parallelization topology to harness the shared and distributed computational power of HPC facilities. Though most of the studies carried out by these codes do not involve any calculation that require shared data between each time step among the MPI threads, but still use MPI topology to deploy the multiple particle trajectory tracing simulation and then at the end of the run, writing and arranging the data in the output files.

PARTICLE-3D code developed initially is one such serial code to study the charge particle dynamics under several combinations of magnetic fields in a tokamak along with the external field perturbations. Previously P3D code was used to trace low energy electron orbit to estimate the error field associated with the toroidal field coils of SST-1 tokamak [16].Recently with more complex magnetic field structure involving ADITYA tokamak [24] plasma equilibrium, the banana orbit of 1 and 3 MeV RE with larger pitch angle 75^o is simulated, see Fig. 1(a). The mirror point from which the particle bounce back also known as the banana orbit tips, often used by such code developers to benchmark their codes with the theoretical estimation of the mirror points. The mirror point obtained from the simulation result shown in Fig. 1(a) is in agreement with the theoretical estimation of the same with in an error of $< 0.15\%$.

Also to understand the RE dynamics in depth, the Poinceré section plot 20 REs placed across the plasma radius with 0^o pitch angle are traced for 60 poloidal transits, as shown in Fig. 1(b). Simulation results reveals that the shrinking of confinement volume is larger in case or 3 MeV REs as compared to the 1 MeV REs. This in turn implies that the each particle of 1 MeV and 3 MeV energy would experience entirely different local magnetic field configuration along their orbit. Therefore each orbit is different from one another and the problem is highly nonlinear in nature.

Simulation results shown in Fig. 1(a) and (b) obtained after lots of computation as the runaway electron orbit tracing involves resolving gyro motion of the charge particle in a complex space dependent magnetic field inside the tokamak. In addition to that with the application of external magnetic fields the drift orbit topology shown in Fig. 1(b) would be drastically changed and the resultant drift orbit dynamics would be responsible for either the prior loss or better confinement of such REs. Hence a large number of such simulations are required to study and understand the performance of the external magnetic field perturbation for different plasma configurations as carried out and presented in [17], for which the required computations

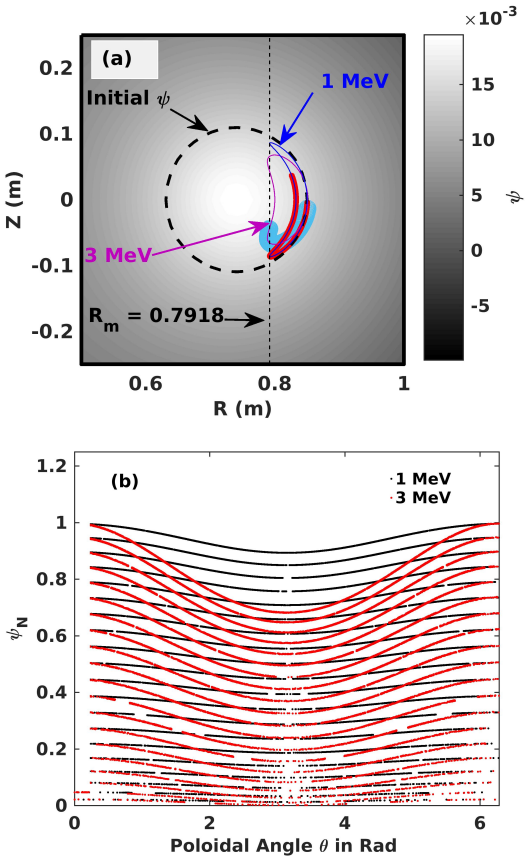

Fig. 1: (a) Banana orbit of 1 and 3 MeV RE in ADITYA plasma equilibrium magnetic field configuration (flux ψ). Poloidal flux surface at the starting point of the particle is marked as Initial ψ. (b) Poinceré section plot of 20 REs in ADITYA plasma equilibrium magnetic field without any external perturbation. Here ψ_N represents the normalized poloidal flux.

would have taken wall time of more than a year to complete. Therefore simulation time needs to be decreased by enhancing the performance of the P3D code through optimization.

B. *Test cases for P3D code performance enhancement*

Considering the highly nonlinear nature of the orbit topologies, appropriate test cases needs to be designed to assess the code performance. Therefore the results discussed in Fig. 1(a) and (b) are considered as four tests to be used for the assessment of *P3D* code optimization and parallelization performances:

- Test-1: Banana Orbit of a 1 MeV RE with 1 poloidal transit.
- Test-2: Banana Orbit of a 3 MeV RE with 1 poloidal transit.
- Test-3: Poinceré section plot for 20 REs of 1 MeV, with 60 poloidal transits.
- Test-4: Poinceré section plot for 20 REs of 3 MeV, with 60 poloidal transits.

Test-1 and Test-2 would serve the purpose of assessment of the performance optimization on a single core while Test-2 and Test-3 would be capable of assessing the overall performance of the *P3D* code without and with the parallelization.

C. *Algorithmic topology of Base version of P3D code*

At the beginning of the development of any code it is of prime importance that the code should give desirable result which can be bench marked against the physical phenomenon for which the code is being developed. The priority of the code performance comes afterwards, after the basic functionalities are developed and benchmark results are obtained. Similarly in this case also the prime functionalities were developed first and the bench mark studies were carried out as described in the previous subsection.
The base version of the P3D code was developed first with the following functionalities and algorithmic topology:

- A main program that would call all other programs via subroutines in order to perform the required simulation requested via input file: TOK-3D
- Locations from where the particles would be launched and initial field would be estimated are prepared on a grid by the GRID-3D code.
- Magnetic fields at the requested locations are estimated by the BGRID-3D code via B3D subroutine that sums up magnetic field from the following sources:
  - Coil gererated magnetic fields are calculated by EFFI code [25].
  - Plasam Equilibrium calculated by IPREQ code [26] are used by BRBZ subroutine to estimated equilibrium field.
  - Magnetic fields from other sources are estimated by the respective BEXT subroutines as required.
- Particle orbits are traced via PARTICLE-3D (P3D) code [17] corresponding to the inputs supplied by all the previous codes: TOK-3D, GRID-3D, BGRID-3D and instantaneous field estimation routine B3D.

Fig. 2 depicts the flow chart of the base version of the serial P3D code. This version of the code is capable of calculating both single particle tracing and multi-particle tracings as the results shown in Fig. 1, depending on the requested simulation set-up.

The base version of the P3D code is capable of performing the required simulations with acceptable accuracy. But the wall time taken by this version of the code for a single run is quite large, which needs to be reduced. Thus both optimization and parallelization is required to enhance the performance of the code, which are discussed in the next section.

V. Optimization and Parallelization of *P3D* Code

It is always recommended that before one goes to make changes in the code for parallel and distributed computation, the code must be portable to any computing system and needs to be optimized for single CPU core run. Now code optimization techniques includes many aspects starting from

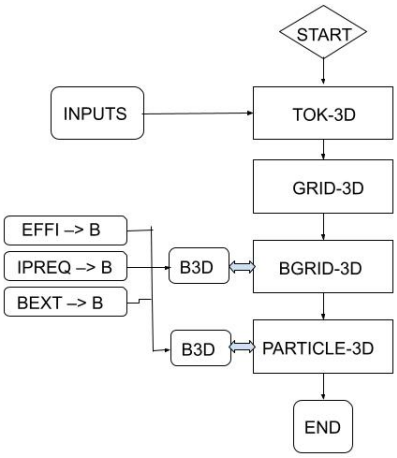

Fig. 2: Flow chart of the base version P3D code in serial configuration.

compilation time reduction, constant and variable propagation, common copied code elimination, dead code elimination to refactorization and restructuring of the code to name a few among others. Out of these optimization techniques in optimizing *P3D* code, we have followed some of them which are discussed in below subsections.

A. *Code portability enhancement*

The *base* version of the *P3D* code was written in *FORTRAN 77* language following the algorithmic topology described in Fig. 2. Parts of code, for example EFFI code in some of its sections uses very old *FORTRAN* syntax that can only be compiled by *Legacy* compiler such as *Lahey/Fujitsu Fortran 95 2004 (LF95)* compiler, therefore the base version of the *P3D* code could only be compiled with the said compiler. Now not all the HPC systems have such commercial legacy compilers in this era of open source compilers, thereby limiting the code portability largely. Therefore to make the code portable across the HPC platforms it needs to be modified to be compiled by modern *Intel IFORT* compiler. With this modification modern code profilers such as *Intel Vtune* Profiler can also be used to identify performace bottle necks and then rectify those accordingly to achieve better performance of the code.

So the parts of the *P3D* code has been modified with the *FORTRAN syntax* compatible with the *Intel IFORT* compiler. With this exercise, higher code portability has been achieved for the *base* version of the *P3D* code. Though the code portability has been enhanced, but the *IFORT* compiled code took $>$4 times larger time than the *LF95* code for the Test-1 and $>$7.5 times larger time for the Test-2. Therefore further optimization was required.

B. *Performance optimization through refactorization*

At this point it is observed that the base version of the code making the executable file very large as it has all the components of the code as discussed in the context of Fig. 2. While running the executable, most of it is not working while tracing the particle through P3D.

Therefore the code has been restructured and refactorized heavily, so that the preprocessing jobs upto the *BGRID-3D* in Fig. 2 can be done independently which gave rise to the independent field calculating code *BGRID-3D*. *BGRID-3D* produces all the output files those are required by the P3D code and the *P3D* code has been converted into another main program to trace the particle orbits once *BGRID-3D* is run before hand. The algorithmic topology becomes very simple for the *P3D* code as shown in Fig. 3(a). Fig. 3(b) shows the functionality of *B3D* algorithm which is used by both *BGRID-3D* and *P3D* independently. This *Modular* version of the code is compilable by both *IFORT* and *LF95* compilers.

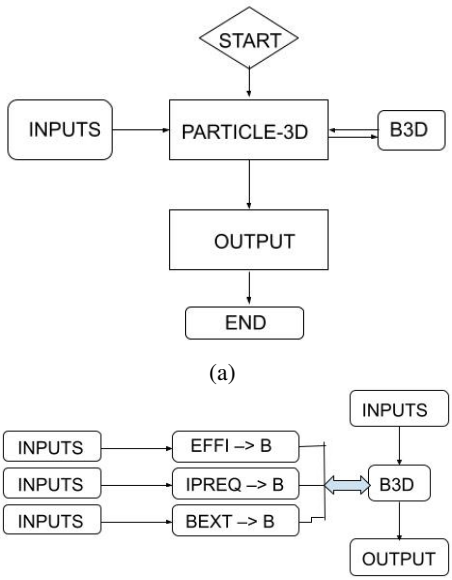

Fig. 3: (a) Flow chart of the refactorized and restructured *P3D* code in *Modular* configuration. (b) Structure of the *B3D* program algorithm that provides the final magnetic field estimation for a particular point. *B3D* code communicates with the other routines, which have their own respective inputs.

The performance of the *IFORT* compiled code in *modular* version increased slightly as compared to the *IFORT* compiled *base* version of the code for both Test-1 and Test-2. Whereas *LF95* compiled *modular* code give performance slightly better than the *LF95* compiled *base* version code for Test-1 and slightly poor performance than the base version for Test-2.

To understand and identify where exactly performance of the code is lagging, the *IFORT* compiled code is profiled by the *IFORT Vtune* profiler for the Test-2. Hotspot analysis Summary by the *IFORT vtune* profiler is shown in Fig. 4. Profiling hotspot analysis shows that two functions, one is "*_read*" linked to one of the compiler module "*libpathread.so.0*" and the other one "*for_read_input*" linked to the *P3D* executable *ip3dexe.out* is taking most of the time of the execution time. It clarifies that reading of data from a input file in the executable

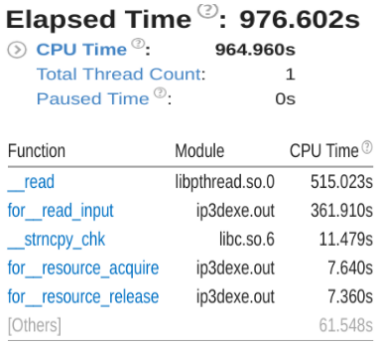

Fig. 4: Hotspot analysis summary from *Intel Vtune* profiler for *P3D* code in *Modular* configuration.

costing huge amount of time, which need to be efficiently managed to reduce the computational time.

C. *Performance optimization through I/O optimization*

As identified by the hotspot analysis by the profiler, an in depth review of the *P3D* code and its subroutines reveals that the *IPREQ* generated plasma equilibrium input file read through *B3D* algorithm via *BRBZ.f* routine was carried out at every time step of the *P3D* particle tracing computation. This

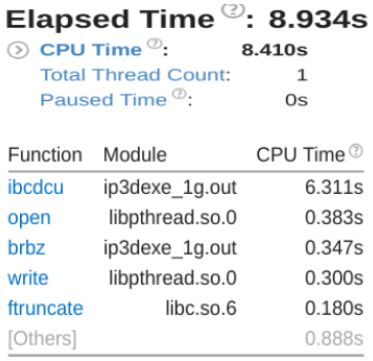

Fig. 5: Hotspot analysis summary from *Intel Vtune* profiler for P3D code in *I/O optimized Modular* configuration.

repeating task of file opening was also making the compiler to perform the "_read" function work continuously. Therefore this problem is addressed by opening the file only once in the main programs i.e. in *BGRID-3D* and *P3D* code and then transferring the data via common block to the required subroutine *BRBZ.f*. This modified I/O optimized version of the code is compilable by both *IFORT* and *LF95* compilers. When this *I/O optimized* version code compiled with *IFORT*, executed for Test-2, it achieves remarkable enhancements in single-core performance. The Hotspot profiling summary for the *I/O optimized* version is shown in Fig. 5. It is found that with the *I/O optimized Modular* version of the code the execution time is reduced drastically for a single core run for the Test-2. Now the code is ready to be converted into parallel code to harness the computational power of the HPC facilities.

D. *Performance enhancement through parallelization*

Now that the performance of the *P3D* code has been optimized and enhanced significantly we further focused on optimizing the code for muti particle runs by exploiting the distributed computational topology offered by the institutes HPC facility *ANTYA*. To parallelize a code on a CPU based HPC facility one can adopt any one of the following paradigm based on the algorithmic topology of the problem:

- **Embarrassingly Parallel (EP) [19], [27], [28]:** used when the parallel threads have no dependency among each other. Such kind of problems are highly parallelizable.
- **OpenMp (OMP):** Shared memory computational topology, parallelizable upto the number of cores available in one node of a HPC facility. Useful when communication is required among the parallel threads during iterations.
- **Message Passing Interface (MPI):** Distributed memory computational topology used when communication is required among the parallel threads running across different nodes of the HPC system.

In our case, parallel threads require no communication among themselves. Initial calculations are performed by a preprocessing code *BGRID-3D*, and post-processing can be done afterwards of the parallel computation of the *P3D* estimated orbits of the individual particles. Thus, OMP or MPI topologies aren't needed for parallelizing the code. Since the required threads exceed single-node cores, OMP isn't suitable for this code. Adapting P3D to be compatible with MPI libraries demands extensive modifications due to its reliance on outdated *legacy Fortran syntaxes* and subroutines. This process would not only be labour-intensive but also introduce complexity into the code. Therefore, the *Embarrassingly Parallel (EP)* topology was chosen, implemented via the Array Job deployment via PBS Job Scheduler on *ANTYA*. For multi-particle runs Test-3 and test-4 are carried out by both *IFORT* and *LF95* compiled code. Performance enhancements achieved are compared in the following section.

VI. Comparative results after optimization

In the previous section the optimization process was started with the *LF95* compiled *base* version *P3D* code and then *modular* version of the code has been obtained. After that the modular version of the code again optimized for I/O operation, that led to the *I/O optimized* version of the *P3D* code. Again all three versions of the codes are compilable by both *legacy LF95* compiler and *modern IFORT* compiler. All these 6 versions of the codes are tested with the 4 test cases mentioned in the previous section. The performance enhancements of each version, after optimization are given by speed up summaries, shown in Fig. 6 to Fig. 9.

To quantify the performance boost obtained for the *P3D* code, we utilize two speed-up formulas [19] based on the wall times (t_w) required by each code version to solve the test problems. These speed-up formulas are outlined below.

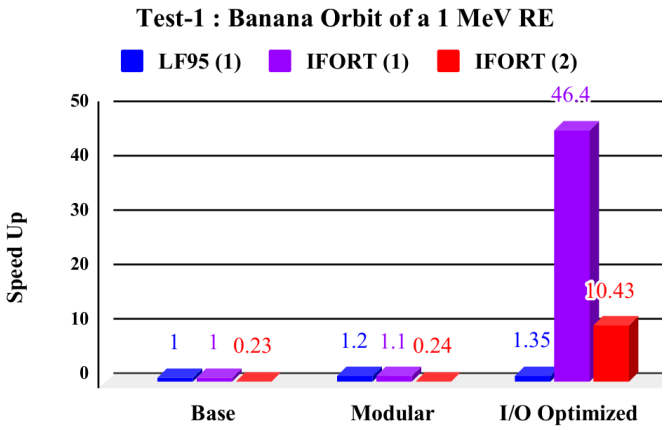

Fig. 6: Speed up achieved for Test-1, when compared to (a) corresponding *base* version of the *LF95* or *IFORT* compiled code, (b) *LF95* compiled *base* version of the code.

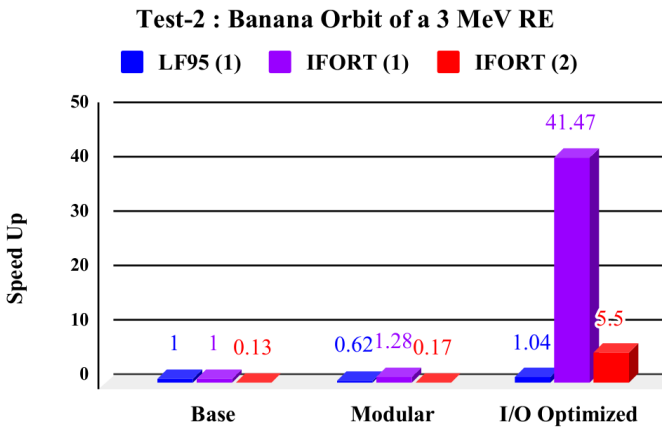

Fig. 7: Speed up achieved for Test-2, when compared to (a) corresponding *base* version of the *LF95* or *IFORT* compiled code, (b) *LF95* compiled *base* version of the code.

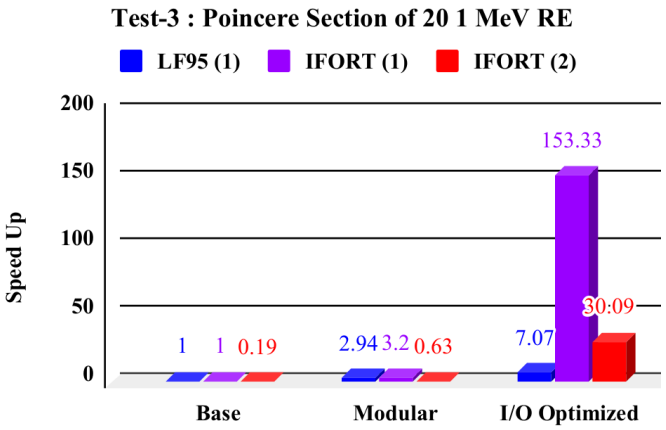

Fig. 8: Speed up achieved for Test-3, when compared to (a) corresponding *base* version of the *LF95* or *IFORT* compiled code, (b) *LF95* compiled *base* version of the code.

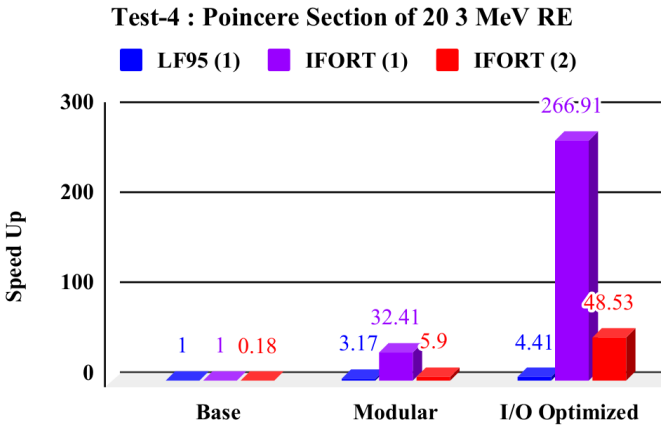

Fig. 9: Speed up achieved for Test-4, when compared to (a) corresponding *base* version of the *LF95* or *IFORT* compiled code, (b) *LF95* compiled *base* version of the code.

- 1. speed up = t_w (version) / t_w (base version) ; here both numerator and denominator version are compiled with same compiler.
- 2. speed up = t_w (version) / t_w (base version) ; here denominator version is compiled with *LF95* compiler.

It can be easily observed from the Fig. 6(a) and (b) that the performance of the *P3D* code has been increased upto 1.35 times when compiled through *LF95* compiler, whereas the speed up of the code is 46.4 times for the *IFORT* compiled code as per the speed up formula - (1). Therefore the final speed up achieved for the Test-1 by formula - (2) is 10.43 as shown in Fig. 6(b). As the 4 different tests are highly nonlinear in nature different final speed up is achieved by the final *I/O optimized modular* version of the code compiled with *IFORT* compiler. Huge performance gains in terms of speed up 30.09 and 48.53 times as shown in Fig. 8(b) and Fig. 9(b) emphasizes the benefit of this *IFORT* compiled *I/O optimized modular* version code along with the *Embarrassingly Parallel* topology achieved through *PBS_Array_INDEX* functionality over the *LF95* compiled *base* version of the code for multi particle tracing.

VII. CONCLUSION

This study employs a hands-on approach to optimise the nuclear fusion energy application code *P3D*. Fig. 6 to Fig. 9, illustrates, that the *legacy LF95* compiled code shows limited improvement over the base version, while the modern *IFORT* compiled code significantly enhances performance along with the code portability across HPC systems.

It is evident from this study that code refactorization and profiling through modern profilers, like *Intel Vtune*, play key roles in optimization. Also understanding the problem's algorithmic topology helps select the appropriate PDC topology, in this case, *Embarrassingly Parallel*, facilitated by Array Job deployment, rather than complex MPI parallelization. The final *IFORT* compiled, modular, I/O optimised version of the P3D code parallelized via EP topology leads to substantial performance improvement as compared to the *legacy* compiler

LF95 compiled base version of the code for all the test cases as summarised in table I.

TABLE I: Performance enhancement ratio of final *Optimised* version over *base* version of the *P3D* code.

Test cases	Speed up
Test-1	10.43
Test-2	5.50
Test-3	30.09
Test-4	48.53

HPC outreach program [17] would arrange HPC workshops to teach and train students, researchers and young scientists in the field of HPC/PDC. Combining with the HPC outreach program, this methodical, step-by-step hands-on approach discussed in this paper for improving performance of an application will help in understanding and learning following concepts for the participants:

- Importance of code portability to improve the usability of the code.
- Importance of suitable compiler selection to enhance the code portability and performance.
- Use and role of code profilers in enhancing the code performance.
- Elements of code optimization to reduce the time of simulation
- Exploitation of HPC system for faster large-scale simulations through PDC and parallelization.

We anticipate that incorporating this practical activity into IPR's HPC outreach workshops will heighten student engagement, enrich their understanding of PDC concepts, and contribute value to the HPC/PDC community.

The authors ensure that the results presented here is reproducible. All the results are made available through GitHub repository: https://github.com/sduttaipr/EduHipc2023_Res.git

ACKNOWLEDGMENTS

The authors would also like to acknowledge the Computer Center of IPR for providing all the necessary resources on *ANTYA* cluster of the institute for this project.

REFERENCES

[1] D. Reed, D. Gannon, and J. Dongarra, "Reinventing high performance computing: Challenges and opportunities," 2022.

[2] A. Papavasiliou, S. S. Oren, and B. Rountree, "Applying high performance computing to transmission-constrained stochastic unit commitment for renewable energy integration," *IEEE Transactions on Power Systems*, vol. 30, no. 3, pp. 1109–1120, 2015.

[3] S. Portegies Zwart, "The ecological impact of high-performance computing in astrophysics," *Nature Astronomy*, vol. 4, no. 9, pp. 819–822, 2020.

[4] R. Kendall, J. C. Carver, D. Fisher, D. Henderson, A. Mark, D. Post, C. E. Rhoades, and S. Squires, "Development of a weather forecasting code: A case study," *IEEE Software*, vol. 25, no. 4, pp. 59–65, 2008.

[5] E. O. Pyzer-Knapp, J. W. Pitera, P. W. Staar, S. Takeda, T. Laino, D. P. Sanders, J. Sexton, J. R. Smith, and A. Curioni, "Accelerating materials discovery using artificial intelligence, high performance computing and robotics," *npj Computational Materials*, vol. 8, no. 1, p. 84, 2022.

[6] A. Mishchenko *et al.*, "Numerical tools for burning plasmas," *Plasma Physics and Controlled Fusion*, vol. 65, no. 6, p. 064001, apr 2023. [Online]. Available: https://dx.doi.org/10.1088/1361-6587/acce68

[7] A. K. Singh, J. Mahapatra, J. Chowdhury, D. Aggarwal, T. Hayward-Schneider, R. Ganesh, E. Lanti, and L. Villard, "Gyrokinetic simulation of short wavelength ion temperature gradient instabilities in the aditya-u tokamak," *Nuclear Fusion*, 2023.

[8] K. L. K. Lee and N. Kumar, "Artificial intelligence for scientific discovery at high-performance computing scales," *Computer*, vol. 56, no. 4, pp. 116–122, 2023.

[9] L. N. Long, P. J. Morris, K. Morooney, and S. Kellogg, "The teaching and learning of high performance computing," *Journal of Engineering Education*, vol. 87, no. S5, pp. 591–597, 1998.

[10] A. J. Kornecki, "Computing curricula for the 21st century," *IEEE Distributed Systems Online*, vol. 9, no. 2, pp. 2–2, 2008.

[11] F. G. Tinetti, M. A. Lopez, P. G. Cajaraville, and D. L. Rodrigues, "Fortran legacy code performance optimization: Sequential and parallel processing with openmp," in *2009 WRI World Congress on Computer Science and Information Engineering*, vol. 2, 2009, pp. 471–475.

[12] B. Chapman, G. Jost, and R. Van Der Pas, *Using OpenMP: portable shared memory parallel programming*. MIT press, 2007.

[13] G. W. Sabot, *High performance computing: problem solving with parallel and vector architectures*. ACM Press/Addison-Wesley Publishing Co., 1995.

[14] R. Suda, H. Takizawa, and S. Hirasawa, "Xevtgen: Fortran code transformer generator for high performance scientific codes," *International Journal of Networking and Computing*, vol. 6, no. 2, pp. 263–289, 2016.

[15] M. Kikuchi, "A review of fusion and tokamak research towards steady-state operation: A jaea contribution," *Energies*, vol. 3, no. 11, pp. 1741–1789, 2010. [Online]. Available: https://www.mdpi.com/1996-1073/3/11/1741

[16] S. Dutta *et al.*, "Numerical simulation and experiment of error field measurement using luminous trace of electron beam in sst-1," *Plasma Science and Technology*, vol. 21, no. 10, p. 105101, 2019.

[17] S. Dutta, D. Sharma, R. L. Tanna, J. Ghosh, R. Srinivasan, and D. Raju, "Deconfinement of runaway electrons by local vertical magnetic field perturbation," *Under Communication*, 2023.

[18] D. Aggarwal, A. Bokshi, and D. Lad, "A hands-on approach for scalable parallel applications development: From testbed to petascale," in *2022 IEEE 29th International Conference on High Performance Computing, Data and Analytics Workshop (HiPCW)*, 2022, pp. 36–43.

[19] N. Wang *et al.*, "The application of nearly embarrassingly parallel computation in the optimization of fluid-film lubrication©," *Tribology transactions*, vol. 47, no. 1, pp. 34–42, 2004.

[20] D. Valentine, "Hpc/pdc immunization in the introductory computer science sequence," in *2014 Workshop on Education for High Performance Computing*, 2014, pp. 9–14.

[21] V. Saravanan, A. Alagan, and K. Naik, "Computational biology as a compelling pedagogical tool in computer science education," *J. Comput. Sci*, vol. 11, no. 1, pp. 45–52, 2020.

[22] D. Aggarwal *et al.*, "Edupar posters," in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2019, pp. 347–349.

[23] G. Papp, M. Drevlak, T. Fülöp, and P. Helander, "Runaway electron drift orbits in magnetostatic perturbed fields," *Nuclear Fusion*, vol. 51, no. 4, p. 043004, 2011.

[24] I. Bandyopadhyay *et al.*, "Modelling of ohmic discharges in aditya tokamak using the tokamak simulation code," *Plasma physics and controlled fusion*, vol. 46, no. 9, p. 1443, 2004.

[25] S. J. Sackett, "Effi: A code for calculating the electromagnetic field, force, and inductance in coil systems of arbitrary geometry," *Unknown*, 1978.

[26] D. Sharma, R. Srinivasan, J. Ghosh, P. Chattopadhyay *et al.*, "Aditya upgradation–equilibrium study," *Fusion Engineering and Design*, vol. 160, p. 111933, 2020.

[27] J.-C. Régin, M. Rezgui, and A. Malapert, "Embarrassingly parallel search," in *International conference on principles and practice of constraint programming*. Springer, 2013, pp. 596–610.

[28] M. Li *et al.*, "Branch-train-merge: Embarrassingly parallel training of expert language models," *arXiv preprint arXiv:2208.03306*, 2022.