January 2021

Volume 12 Issue 1

JoCsE

Journal Of Computational Science Education

Promoting the Use of Computational Science Through Education

ISSN 2153-4136 (online)

Volume 12, Issue 1 **January 2021**

JOCSE

Journal Of Computational Science Education

The Journal Of Computational Science Education (JOCSE), ISSN 2153-4136, published in online form, is a supported publication of the Shodor Education Foundation Incorporated. Materials accepted by JOCSE will be hosted on the JOCSE website, and will be catalogued by the Computational Science Education Reference Desk (CSERD) for inclusion in the National Science Digital Library (NSDL).

Subscription: JOCSE is a freely available online peer-reviewed publication which can be accessed at `http://jocse.org`.

Contents

Introduction to Volume 12 Issue 1

Steven I. Gordon
Editor
The Ohio State University
Columbus, OH
gordon.1@osu.edu

FOREWORD

This issue of the journal contains two major articles and five articles summarizing the research experiences of students. Toolin, Dion, and Erickson describe a newly created computer science licensure program for preservice teachers. They review the need for computer science courses in the curriculum and various efforts to license teachers. They then summarize the work of their collaborative at the University of Vermont to create and implement a CS licensure program for preservice teachers.

The paper by Sharma presents an end to end data science application where students in an introduction to scientific computing class apply machine learning to identify and classify images of chemistry laboratory glassware. Students collected pictures of a variety of glassware and then used several neural network applications in the Wolfram language to classify the pictures. This machine learning example helped students to understand the potential use of such techniques across their STEM disciplines.

Rivera and Araya summarize an analysis of the turbulent boundary layers related to supersonic flight. They used computational fluid dynamics software to model several of the related boundary flow conditions for this system.

Adewale compares the performance of serial and parallel ray tracing techniques. He used OpenMP with C++ to implement a parallel version of the ray tracing routines that was 10 times faster than the serial version.

In their article, Barragan and Groves describe the use of machine learning with a neural network to evaluate the energies in chemical structures. They compare several approaches to using the neural network that help to improve its predictive accuracy.

Noneman et al. use molecular dynamics and Monte Carlo simulations to explore the behavior of buckminsterfullerene in water. They compare seven models for the self-assembly of this molecule.

The final student article by Kaman, Edwards, and McGarigal compares several models of turbulent mixing of two fluids. They explore the runtime behavior of the codes using MPI and a hybrid MPI, OpenMP parallel programming model.

The Computer Science Education Collaborative: Promoting Computer Science Teacher Education Programs for Preservice and In-service Teachers

Regina Toolin
University of Vermont
Burlington, VT
Regina.Toolin@uvm.edu

Lisa Dion
University of Vermont
Burlington, VT
Lisa.Dion@uvm.edu

Robert Erickson
University of Vermont
Burlington, VT
Robert.Erickson@uvm.edu

ABSTRACT

This article reports on the efforts of the Computer Science Education Collaborative during the period between 2018–2020 to develop and implement a new computer science licensure program for preservice teachers seeking a license to teach computer science in grades 7–12 in Vermont. We present a brief review of the literature related to computer science teacher education and describe the process of developing the computer science education minor and major concentration at the University of Vermont. As a form of reflection, we discuss the program development process and lessons learned by the collaborative that might be informative to other institutes of higher education involved in CS teacher education program design and implementation. Finally, we describe next steps for developing in-service licensure programs for teachers seeking computer science professional development or licensure in grades 7–12.

Keywords

Computer Science Education, Computer Science Standards, Rural Education, Micro-credentials, In-service and Preservice Teachers

1. INTRODUCTION

Despite national efforts to highlight Computer Science (CS) education and careers in the U.S., a comparatively small percentage of students pursue CS postsecondary degrees and careers [1]. Computing represents two-thirds of projected new STEM jobs in the U.S., but only 4% of college students overall earn a CS degree [13]. Computing and information technologies (IT) have driven many aspects of Vermont's economic growth, as evidenced by the presence of Dealer.Com, NRG Systems, Competitive Computing, IBM, Global Foundries, the Vermont Technology Alliance, and over 200 other related companies statewide. Vermont's IT future is bright; however, there is a gap between CS employment opportunities and CS learning opportunities available for K–12 students and teachers in the state.

Opportunities to learn CS in Vermont demonstrate some signs of improvement over the past few years. In 2020, 60% of VT high schools reported offering CS courses as part of the curriculum [7]. However, when data are disaggregated for socioeconomic status, ethnicity, and gender, only 50% of rural high schools in VT offer CS courses.

Opportunities for underrepresented and underserved students to engage in CS learning have also been limited in that 28% are female, 7% Latinx, 9% Asian, and no African American or Native American students took AP CS exams during this period [7].

Over 62% of K–12 principals in VT believe CS should be a required core class; however, limited funds for hiring and training teachers is cited as the primary barrier for offering CS courses at their schools (Code.org, 2019). In 2019, a state-wide survey of teachers (n=270) conducted by the VT Agency of Education (AOE) found that 51% of respondents engaged students in some form of CS activities including Code.org Hour of Code, Lego robotics, makerspace, or other coding activities. Approximately 20% reported having after-school computer clubs at their school, and 27% engaged in CS professional development. In 2017, approximately 200 new secondary teachers were prepared to teach CS in the US [12]; however, no teachers graduated from a Vermont institute of higher education (IHE) prepared to teach CS during this period [12, 6]. To date, only 27 out of approximately 8,000 teachers are licensed to teach CS in VT.

In 2018, the Computer Science Education Collaborative (CSEC) was established by the University of Vermont's (UVM) College of Education and Social Services (CESS) and College of Engineering and Mathematical Sciences (CEMS) and the Vermont Agency of Education (VT AOE) with the primary mission to address the gaps and inequities in CS learning opportunities for students and teachers in Vermont. The CSEC prioritized the development of a new CS licensure program for preservice teachers seeking an initial CS licensure endorsement in grades 7–12. Following a comprehensive review process, the CS licensure program was approved by the UVM Faculty Senate and Board of Trustees in May 2019 followed by approval from the VT Agency of Education in May 2020.

This article reports on the efforts of the CSEC to develop and implement a new computer science licensure program for preservice teachers seeking a license to teach CS in grades 7–12. What follows is a brief review of pertinent literature related to computer science teacher education in the US, a description of the computer science education minor and major concentration at UVM, a reflection on the program development process, and lessons learned by the collaborative that might be informative to other IHEs involved in CS teacher education program design and implementation. Finally, we offer some insights into next steps for the development of in-service licensure programs for teachers seeking CS professional licensure in grades 7–12.

DOI: https://doi.org/10.22369/issn.2153-4136/12/1/1

2. RESEARCH ON COMPUTER SCIENCE TEACHER EDUCATION

A number of CS professional education programs and pathways are now available for teachers to acquire the knowledge, skills and practices leading to certification to teach CS in K–12 environments across the US [4]. Currently, three distinct types of CS teacher certification pathways exist including subject, course-specific, and local education agency (LEA) determined pathways. States with subject certifications require the same teacher certification for any CS course taught. Course-specific states have different requirements depending on the specific CS course teachers plan to teach. LEA states have no state-wide certification requirements for teaching CS courses; an LEA sets these requirements for their district.

With limited pathways to CS certification in the US, Code.org recommends that LEAs allow teachers who currently teach CS or who have some CS professional learning experiences to teach CS under a temporary license. In this pathway, teachers may pursue an "add-on endorsement" that includes CS methods and content requirements to be earned, possibly through a certification assessment or 12 credit hours of coursework. Code.org also suggests the development of a full certification pathway that includes CS content courses, general education coursework, and computer science methods and practice teaching [4].

In Utah, a course-specific state, different levels of endorsement pathways provide less intimidating entry points for teachers without a CS background [4]. To earn a credential to teach Exploring Computer Science, teachers must complete Code.org CS Fundamentals, Teaching Methods (ECS PD), and an Industry test (Certiport IC3). A CS Level 1 endorsement allows teachers to teach AP Computer Science Principles and Computer Programming 1 and 2 upon completion of Teaching Methods (ECS, CSP), 3 CS courses (CS50, Oracle, Bottega), and/or an Industry test (MTA, Oracle). A CS Level 2 endorsement requires Teaching Methods (ECS, CSP), 5 CS courses (includes the 3 for the CS Level 1 endorsement), and an Industry test (MTA, Oracle).

Micro-credentials provide a competency-based pathway to certification that recognizes a teacher's existing expertise and increases accessibility by making certification job-based and less costly [5]. Micro-credentials are performance-based, allowing for teachers to demonstrate competency in content and pedagogy and can be "stacked" together in order to meet endorsement requirements. For example, Arizona allows teachers to be licensed through 12 credit hours of course work (for grade 6–12 CS endorsement) and requires fifteen hours of professional learning or an analogous micro-credential to be equivalent to one credit hour [5]. Micro-credentialing opportunities are provided through programs like BloomBoard CS Micro-credentials that facilitate the earning of micro-credentials: for example, "understanding cultural differences" and "using formative assessment to modify future instruction" [2].

Currently, Vermont is a subject certification state where preservice teachers must meet state computer science endorsement standards and the equivalent of a CS minor to teach CS in grades 7–12. In addition, preservice teachers must successfully complete a practicum and student teaching experience, required education coursework (e.g. foundations, diversity, curriculum, special education, educational technology, and CS methods courses) and the VT Licensure Portfolio (VLP) to earn licensure in grades 7–12 computer science education.

3. COMPUTER SCIENCE EDUCATION PROGRAM DESCRIPTION

The University of Vermont (UVM) has an enrollment of 10,612 undergraduates, 1,552 graduate students, and 466 medical students with 1,685 full-time and part-time faculty and offers students choices for programs from among more than 100 undergraduate majors, 52 masters, and 25 doctorate degrees. UVM is the only research university in the state, the state's only Carnegie-classified Research-Extensive institution, and a leader in research expenditures among public universities with enrollments less than 15,000. Specialized computer science facilities include a new STEM complex, Vermont Advanced Computing Core (VACC), and the Vermont Complex Systems Center. These unique distinctions make UVM an ideal academic setting to support computer science education on its campus.

The College of Engineering and Mathematical Sciences (CEMS) houses the Department of Computer Science and has an enrollment of 400 undergraduates among BS degrees in Computer Science, Computer Science & Information Systems, and Data Science, as well as a BA degree in Computer Science offered through the College of Arts and Sciences (CAS). Over 60 graduate students are enrolled in the Computer Science and Complex Systems & Data Science degrees. The College of Education and Social Services (CESS) has the only nationally recognized Council for the Accreditation of Educator Preparation (CAEP) teacher preparation programs in Vermont. Faculty and staff work with 825 undergraduate and 370 graduate students in more than 30 programs housed in the Education, Leadership, and Social Work departments. The Secondary Education Program has an enrollment of 150 students who are required to meet all VT AOE Endorsement Standards and Vermont Licensure Portfolio (VLP) requirements for certification in grades 7–12. The new CS minor and major concentration are fully approved licensure programs for preservice teachers interested in teaching computer science in grades 7–12 in the Secondary Education Program.

3.1 UVM's Computer Science Education Minor and Major Concentration

In September 2018, the Computer Science Education Collaborative (CSEC), comprised of 2 faculty and 1 staff from the Computer Science Department, 3 faculty from the Department of Education, and the VT AOE State Director of Technology, met to begin the development of a new minor and major concentration in computer science education. This initiative was inspired by previous conversations between education faculty and the VT State Director regarding the recent revision of the Vermont Agency of Education Computer Science Endorsement Standards [15] as well the need for CS learning opportunities for students and teachers across Vermont. In order to advance its goals and objectives, the CSEC submitted and secured an internal STEM Education grant to support CS curriculum development work, CS professional development and a CS Summit for teachers hosted at UVM in August 2019. Early on in the collaborative's efforts, the CSEC bi-weekly meetings focused on the alignment of the VT AOE computer science endorsement standards with existing undergraduate CS and Education coursework. This process revealed that all CS endorsement standards were met by existing coursework with the exception of a CS content specific methods

Table 1. VT AOE CS Endorsement Standards Grades 7–12

Computer Science Knowledge Standards *Adapted from VT AOE CS Endorsement Standards (2019)*
Historical Context — The candidate recognizes the historical development and contributions of individuals or groups, particularly from underrepresented populations related to CS.
Algorithmic Thinking — The candidate can demonstrate algorithmic problem-solving knowledge and skills, design solutions (e.g., problem statement and exploration, examination of sample instances, design, implementing solutions, testing, evaluation, revising).
Computing Systems — The candidate understands computing systems including networks, operating systems, hardware, software and the role of compilers and interpreters in translating languages into machine instruction
Networks and The Internet — The candidate can demonstrate a knowledge of types of networks, internet protocols, the relationship between clients and servers, and cybersecurity.
Data Analysis — The candidate can analyze data by collecting, aggregating, cleaning, and modeling the data through simulations, visualizations, and statistical models.
Algorithms and Programming — The candidate is fluent in at least one high-level language used in current pedagogy and can compare high-level languages, particularly object-oriented program design. This includes designing, implementing, testing, and debugging. Includes an understanding of problem-solving strategies, algorithm analysis, programming concepts.
Impacts of Computing — The candidate understands digital, ethical and intellectual property issues. Candidate is considerate of equitable use of technology for all, knows good digital citizenship, can identify and avoid online threats.
CS Sub-Disciplines — The candidate knows concepts, vocabulary, and issues in two or more CS sub-disciplines (abstract data types, advanced CS algorithms, computer architecture, network/data communications, physical computing, digital forensics, machine learning).
Performance Standards: The candidate demonstrates pedagogical understanding with teaching CS content. Includes planning instruction with problem-solving, current technology, verbal and written communication skills, cooperative learning, and both visual and active activities.
Additional Requirements: A minor in CS, or the equivalent, in undergraduate or graduate coursework.

course that was subsequently developed in collaboration by the CSEC faculty and further described in Section 3.2 of this article. Submission of the CS education minor and major concentration proposals to college and university-wide Curricular Affairs Committees (CAC) occurred in October 2018 followed by approval by both the UVM Faculty Senate and Board of Trustees in May 2019.

This important milestone paved way for the VT Agency of Education Results Oriented Program Approval (ROPA) [16] process that grants institutes of higher education in Vermont the authority to recommend students for teaching licensure in specific disciplines. This process entailed an additional year of data gathering, reporting, proposal submission, and final ROPA approval in May 2020.

The new CSE minor and major concentration reflect the essential knowledge and skills that computer literate students and teachers need to communicate and interact in today's world. All CSE courses are aligned to the VT AOE CS Endorsement Standards (See Table 1) and are offered through the Departments of Computer Science and Education.

Each of the required CSE content courses emphasizes a variety of attributes of computational and abstract thinking skills that also includes a required calculus class. Enrollment in the CSE preservice teacher program began in Fall 2020 with the initial expectation of modest interest from students primarily in the secondary education program seeking initial and/or dual licensure or computer science majors seeking a double major in CS and secondary education or a Master of Arts in Teaching as an accelerated master's student. The required CSE courses for the minor include 5 CS courses and 1 CS methods course, and 9 CS courses plus the equivalent of an education major for the CS major concentration (See Table 2).

Table 2. CSE Minor (19 Credits) and Major Concentration (31 Credits) Requirements

CS 008: Introduction to Web Design — 3 credits
CS 021: Introductory Programming — 3 credits
CS 064: Discrete Structures — 3 credits **(major only)**
CS 087: Introduction to Data Science — 3 credits
CS 110: Intermediate Programming — 4 credits (Prerequisite: CS 021)
CS 121: Computer Organization — 3 credits (Prerequisite: CS 110)
CS 124: Data Structures and Algorithms (Prerequisites: Math 021, CS 064, CS 110) — 3 credits **(major only)**
CS 166: Cybersecurity Principles — 3 credits **(major only)**
CS 292: Senior Seminar — 1 credit **(major only)**
CS 091: Instructing in Computer Science (recommended)
Math 021: Calculus 1 — 4 Credits
EDSC 237: Teaching Computer Science in Secondary School — 3 credits

All content courses listed as requirements for the CSE minor and major concentration are currently offered through the CS and Mathematics departments. The methods course is offered through the Education department. Secondary education students enrolled in the CS major concentration will complete all 3 phases of the secondary education program including content requirements, university general education and diversity requirements and professional education requirements. See Table 3 for a summary of these requirements.

Table 3. Secondary Education Professional Education Requirements — 46 Credits

EDSP 005: Issues Affecting Persons with Disabilities — 3 Cr.
EDTE 001: Teaching to Make Difference — 3 Cr.
EDFS 002: School and Society — 3 Cr.
ECLD 056: Language, Policy, Race and Schools — 3 Cr.
EDSC 011: Ed. Technology in Secondary Classes — 3 Cr.
EDSC 207: Adolescent Development — 3Cr.
EDSC 209: Practicum in Teaching — 4 Cr.
EDSC 215: Teaching Reading in Secondary Schools — 3 Cr.
EDSC 216: Curriculum, Instruction & Assessment — 3 Cr.
EDSC 237: Teaching CS in Secondary School — 3 Cr.
EDSC 226: Student Teaching Internship — 12 Cr.
EDSC 230: Teaching for Results — 3 Cr.

3.2 Teaching Computer Science in Secondary School

Teaching Computer Science in Secondary School (EDSC 237), a "methods" course developed in collaboration with CS and Education faculty, is required for students enrolled in both the CSE minor and major concentration. This course explores multiple theories and practices of teaching, learning, and assessing computer science in middle school and high school and is aligned to both the Computer Science Teachers Association (CSTA) K–12 Standards [8] and the VT AOE Computer Science Education Endorsement Standards [15]. The course places an emphasis on modeling critical dialogue and reflection about such topics as the nature of computer science; the structure of computer science disciplines; computer science learning standards; best practices of teaching and assessing computer science; and social, legal, ethical, and cybersecurity issues in computer science and computer science education.

EDSC 237 is a capstone course that students take near the completion of CSE program requirements. EDSC 216 — Curriculum, Instruction and Assessment is a prerequisite for EDSC 237 and provides students with the opportunity to develop and apply fundamental knowledge and skills of lesson, assessment, and unit development to their emergent teaching practice. EDSC 237 builds on these pedagogical skills and provides students with multiple opportunities to develop and pilot a variety of CS lessons in a "critical friends" environment with their peers.

EDSC 237 is participatory in nature and practice. Each week, the instructors and students model and present problem-based activities and place-based lessons and projects aligned to the CSTA learning standards and other essential practices necessary to become a master teacher of computer science. Problem-based learning is driven by challenging, open-ended problems with no one right answer. Students work as self-directed, active investigators and problem-solvers in small collaborative groups [10]. Project-based learning is a teaching method in which students gain knowledge and skills by working to investigate and respond to engaging and complex questions, problems, or challenges through scientific and computational thinking over an extended period of time [3]. Place-based education is the process of using the local community and environment as a starting point to teach hands-on, real-world learning experiences in computer science, language arts, mathematics, social studies, science, and other subjects across the curriculum. The goal of place-based learning in the context of teaching computer science is to help students integrate CS principles and practices into community-based projects while simultaneously enhancing students' appreciation for the natural world and creating a heightened commitment to serving as active, contributing citizens [11].

One of the primary outcomes of EDSC 237 is the development of a computer science resource portfolio or collection of related lessons, activities, projects, and assessments that preservice teachers can implement during student teaching. Students are required to align their lessons and activities to the CSTA learning standards and integrate the principles and practices of problem-based, project-based, and place-based principles and practices into their CS resource portfolio. For more information about the methods course, please see the EDSC 237 syllabus [14].

4. LESSONS LEARNED FROM THE COMPUTER SCIENCE EDUCATION COLLABORATIVE

In order to assess the overall progress of the CSEC's work over the past two years, it was beneficial to utilize the evaluation framework of effective research-practitioner partnerships (RPP) developed by Henrick et al. [9]. In particular, our program assessment focused on the following questions: 1.) How did the CSEC work to develop trust and ensure that there was equitable participation among partners? 2.) How did the CSEC support the goals of the participating practitioners? 3.) How did the CSEC learn together? 4.) How will the CSEC ensure that the curriculum work that they have engaged in over the past two years will be fruitful?

4.1 How did the CSEC develop trust and ensure there was equitable participation among partners?

Initially, CSEC members consisted of 2 CS faculty, 1 CS outreach coordinator, 3 education faculty, and the state director of technology. After CSE program approval in year 1, a CS teacher from a local school district joined the partnership. In order to keep the momentum moving forward, this diverse group of stakeholders made a commitment to attend and actively participate (in-person and virtually) in bi-weekly meetings for the purpose of accomplishing our CSE program development and implementation goals. We shared the workload by taking on various research, survey development, and standards alignment tasks. We established meeting norms that encouraged equal voice and input from all partners and invited participation from those who at times seemed reluctant to speak.

4.2 How did the CSEC support the goals of the participating practitioners?

Two of the CSEC partners were school practitioners who had fixed schedules with limited availability for meeting. As a collaborative,

we agreed to conduct our meetings during times that were conducive to practitioners' schedules. As a result, we would often meet during lunch or after school hours. The collaborative made a genuine effort to better understand the roles and responsibilities of K–12 classroom teachers as well as the overall Grade 7–12 school culture and environment. The CSEC was particularly interested in the experiences of one of our partners, a CS classroom teacher, who recently earned CS licensure endorsement through the VT AOE peer review process. Her testimonial informed the development of the CS methods course and was instrumental in conversations pertaining to next steps in developing CS licensure pathways for in-service teachers.

4.3 How did the partnership learn together?

As members of the CSEC, we recognized that each of us brought distinct knowledge and expertise that could advance both the project and the CSEC on the whole. CS faculty contributed in-depth knowledge of CS principles, processes, and practices aligned to VT CS endorsement standards and organized this data into a comprehensive spreadsheet that was invaluable to our program and course development work going forward. Education faculty and practitioners were instrumental in translating the curricular and pedagogical elements of the VT CS endorsement standards as well as in making recommendations for how topics such as equity, ethics, and cybersecurity might make their way into the CS methods course. In addition, on numerous occasions throughout the past two years, CSEC partners attended and presented at international, national, and regional CS conferences. In November 2019, four of the CSEC partners participated in a National Science Foundation CS for All workshop in DC to begin work on a CS for All grant proposal with the intent to develop and support CS pathways for VT in-service teachers.

4.4 How will the CSEC ensure that the curriculum work that they engaged will be fruitful?

After two years of collaborative work and the approval of the CSE minor and major at the university and state levels, we have met our targeted goals. Currently, the role of the CSEC has shifted from program development to program implementation, marketing, and recruitment. With the support of university administrators including chairs, deans, and communications directors, the CSEC is now focused on broadly advertising the CSE program via college websites, course catalogs, and program checklists for students. In addition, CS and Education faculty have been instrumental in directly communicating the progress and outcomes as well as new CSE program information to their departmental colleagues.

5. NEXT STEPS: WHERE DO WE GO FROM HERE?

Through the ongoing collective efforts of the CSEC, it became increasingly evident that the development of a preservice computer science education program was truly only the "tip of the iceberg" regarding the work that still needs to be accomplished in computer science education in Vermont. Given that only 27 teachers are currently licensed to teach CS in Vermont, word of the CSE program quickly spread to in-service teachers who were interested in earning licensure in computer science education. State-wide surveys conducted by the VT AOE and evaluations from a CS Summit hosted by the CSEC in 2019 confirmed our hunch that many teachers (over 300) were interested in CS professional learning opportunities and licensure. Teachers inquired as to whether the CSE minor or major concentration programs would be appropriate to their needs. In some cases, the answer was "yes," but in reality, the preservice CSE program was designed primarily for undergraduate or MAT graduate students who were interested in an initial CS licensure program.

This perceived need has prompted the CSEC to move forward in the development of flexible pathways for in-service teachers that recognize the limitations and constraints of teachers' professional and personal lives and offer alternative options for teachers to earn licensure in computer science. In addition to the traditional undergraduate program described in this article, the CSEC is currently working on 2 additional pathways: a CSE Certificate Pathway and a CSE Individually Designed Pathway for licensed teachers interested in computer science licensure. For the CSE Certificate pathway, in-service teachers would enroll in a blend of CS content and pedagogy courses aligned to the CS AOE endorsement standards that explore best curriculum and teaching practices. Courses would be offered primarily online with options for face-to-face workshops, seminars (synchronous and asynchronous), and field/practicum experiences offered during summer months. This pathway is designed for licensed teachers, as well as others seeking licensure endorsement and methods to integrate CS principles and practices into the 7–12 curriculum.

The CSE Individually Designed pathway is intended for teachers who may already have earned a number of required CSE credits and/or have participated in CS professional development workshops over time or who are new to CS principles and teaching. In this pathway, teachers would work with an advisor to curate a sequence of coursework, workshops (e.g. Code.org), as well as formal and informal CS teaching experiences that demonstrate evidence for meeting AOE CS endorsement standards leading to licensure. Teachers in this pathway would need to apply for the VT AOE peer review process and complete a portfolio that demonstrates how their courses and experiences meet AOE licensure endorsement standards. The development of these flexible pathways will be the primary focus of the work of the CSEC going forward.

The variety of CS professional learning opportunities described in this article have been designed to build preservice and in-service teachers' capacity to offer more authentic, engaging, and inquiry-based CS education to their students. As Vermont and other states increasingly aim to provide high-quality CS education for all students, the Computer Science Education Collaborative through its efforts will continue to advance and support these initiatives over time.

6. ACKNOWLEDGEMENTS

The CSEC was funded by the University of Vermont STEM Education grant in order to support the development and implementation of the computer science minor and major concentration.

The authors would like to acknowledge Peter Drescher, Juniper Lovato, Heather Rogers, Katie Shepherd, and Alan Tinkler for their contributions to the CSEC over the past two years.

7. REFERENCES

[1] Robert D. Atkinson and Merrilea Mayo. 2010. *Refueling the U.S. Innovation Economy: Fresh Approaches to Science, Technology, Engineering and Mathematics (STEM) Education*. The Information Technology & Innovation Foundation. Washington, D.C. Retrieved from https://itif.org/files/2010-refueling-innovation-economy.pdf.

[2] BloomBoard, Inc. What Are Micro-credentials? Retrieved from https://bloomboard.com/what-are-microcredentials/.

[3] Buck Institute for Education. PBLWorks. Retrieved from https://www.pblworks.org/.

[4] Code.org. 2018. *Everyone and No One Can Teach CS: Certifications Eligible to Teach CS (by State)*. Retrieved from docs.google.com/document/d/1XA1zIjuR222BtRWTljNatCD6TYFAZan8eDWqZneG6I8.

[5] Code.org Advocacy Coalition. 2019. *Micro-credentials: A Pathway for Certification and Professional Learning.* Retrieved from https://advocacy.code.org/micro-credentials.pdf.

[6] Code.org Advocacy Coalition and Computer Science Teachers Association (CSTA). 2018. *2018 State of Computer Science Education: Policy and Implementation*. Retrieved from https://code.org/files/2018_state_of_cs.pdf.

[7] Code.org Advocacy Coalition, Computer Science Teachers Association (CSTA), and Expanding Computing Education Pathways (ECEP) Alliance. 2020. *2020 State of Computer Science Education: Illuminating Disparities*. Retrieved from https://advocacy.code.org/2020_state_of_cs.pdf.

[8] Computer Science Teachers Association (CSTA) K-12 Standards. Retrieved from https://csteachers.org/page/about-csta-s-k-12-nbsp-standards.

[9] Erin C. Henrick, Paul Cobb, William R. Penuel, Kara Jackson, and Tiffany Clark. 2017. *Assessing Research-Practice Partnerships: Five Dimensions of Effectiveness*. William T. Grant Foundation, New York, NY. Retrieved from https://wtgrantfoundation.org/library/uploads/2017/10/Assessing-Research-Practice-Partnerships.pdf.

[10] Linda B. Nilson. 2010. *Teaching at Its Best: A Research-Based Resource for College Instructors* (2nd ed.). Jossey-Bass, San Francisco, CA.

[11] David Sobel. 2004. Place-Based Education: Connecting Classrooms and Communities. Orion, Great Barrington, MA.

[12] United States Department of Education. 2019. Title II Higher Education Act: 2018 All States Report Data File. Retrieved from https://title2.ed.gov/Public/DataTools/2018/AllStates.xls.

[13] United States Department of Education, National Center for Education Statistics, Integrated Postsecondary Education Data System (IPEDS), Fall 2001 through Fall 2018, Completions component.

[14] The University of Vermont. Secondary Education Program: EDSC 237: Teaching Computer Science in Secondary Schools. Syllabus. Retrieved from https://drive.google.com/file/d/1AIaAA3ACUrKA3um_F1z4kvr6wtAGbrkd/view

[15] The Vermont Standards Board for Professional Educators. *Rules Governing the Licensing of Educators and the Preparation of Educational Professionals*. Vermont Agency of Education Educator Quality Division. Retrieved from https://education.vermont.gov/sites/aoe/files/documents/Rules%20Governing%20the%20Licensing%20of%20Educators_9_20_2019.pdf

[16] The Vermont Standards Board for Professional Educators. *Results Oriented Program Approval (ROPA) Handbook*. Vermont Agency of Education Educator Quality Division. Retrieved from https://education.vermont.gov/sites/aoe/files/documents/edu-educator-quality-ropa-handbook.pdf.

Laboratory Glassware Identification: Supervised Machine Learning Example for Science Students

Arun K. Sharma
Department of Chemistry and Physics
Wagner College
Staten Island, NY
aksharma@wagner.edu

ABSTRACT

This paper provides a supervised machine learning example to identify laboratory glassware. This project was implemented in an Introduction to Scientific Computing course for first-year students at our institution. The goal of the exercise was to present a typical machine learning task in the context of a chemistry laboratory to engage students with computing and its applications to scientific projects. This is an end-to-end data science experience with students creating the dataset, training a neural network, and analyzing the performance of the trained network. The students collected pictures of various glassware in a chemistry laboratory. Four pre-trained neural networks, Inception-V1, Inception-V3, ResNet-50, and ResNet-101 were trained to distinguish between the objects in the pictures. The Wolfram Language was used to carry out the training of neural networks and testing the performance of the classifier. The students received hands-on training in the Wolfram Language and an elementary introduction to image classification tasks in the machine learning domain. Students enjoyed the introduction to machine learning applications and the hands-on experience of building and testing an image classifier to identify laboratory equipment.

KEYWORDS

Machine Learning, Object Identification, Laboratory Glassware, First-year

1 INTRODUCTION

Machine learning applications are increasingly common in the day-to-day interactions of students with technology. An increasing number of products from thermostats to recommendations for the next TV series or movie to watch use some form of machine learning to augment the user experience. Self-driving cars [1], victory in the game of Go over humans [28], and image classification [7] are some of the more high-profile applications of machine learning. However, in addition to these, such tools are also used in email spam filtering [6], credit score determination [9], as well as many others. An interactive history of machine learning, including references and major applications, has been developed by Google [4]. A recently released review article provides more detailed information on applications of machine learning to scientific domains and specifically to the area of material science research [24].

A variety of resources in the domain of machine learning, neural networks, and their applications is now available online that is accessible to people with a range of technical skills. A full review of machine learning is beyond the scope of this paper. However, the interested reader is referred to multiple freely available resources for additional information and background. Coursera hosts a very popular course on machine learning [11]. Wolfram Research provides multiple training videos and courses to get users started on machine learning basics [15, 17], image classification [20], and many more applications using the Wolfram Language [16]. Google also provides a course for developers to introduce them to machine learning, and the examples are accessible to beginners as well as those with more advanced skills [5].

Supervised learning corresponds to the family of approaches that train a neural network to learn from a training set of labeled examples. The trained network, after testing, is utilized in performing the specialized task on new samples of unlabeled data. Deep learning, based on multi-layer neural networks, has recently outperformed traditional approaches in computer vision and natural language processing. One of the major success stories of deep learning applications is image classification [12]. The goal in image classification is to classify a picture according to a set of possible categories. Transfer learning in the field of computer vision enables the construction and implementation of accurate models rapidly and without rebuilding the entire neural network architecture. In practice, a pre-trained model is adopted that was trained on a large benchmark dataset to solve a problem similar to the one under consideration. Such pre-built models are imported from published literature and then adapted for application to the problem of interest. A comprehensive review of the performance of pre-trained models for computer vision problems using the ImageNet data [23] challenge is provided [2].

A commonly implemented first example in image classification is that of distinguishing images of cats from dogs. A pre-trained neural network is provided with a labeled training set of images. The training is performed, and the trained network's performance is then tested using images that were not part of the training set. The success of training becomes quite evident with the results and can be measured in terms of accuracy of classification. The exercise is quite easy to construct and provides a good first example for students. Another exercise that is widely used is the identification of hand-written digits. The Modified National Institute of Standards and Technology (MNIST) database of hand-written digits

https://doi.org/10.22369/issn.2153-4136/12/1/2

and its classification and identification are also widely used for assessment of potential image classification algorithms [10]. That database includes a training set of 60,000 examples and a test set of 10,000 examples and is also used quite commonly as one of the first examples in this domain.

Our main goals were to introduce students to machine learning applications, highlight the ease of creating such applications using the Wolfram Language, and encourage students to think about possibilities of applying such developments to scientific domains. This project was carried out with students in the author's "Introduction to Scientific Computing" course. The course philosophy and design have been previously described in this journal [26]. That course provides students with an introduction to programming in the Wolfram Language using the Mathematica notebook interface. The crux of the course is to provide students with hands-on experience in production, visualization, and analysis of technical data. Modifications of that course design were also successfully implemented to incorporate a course based undergraduate research style experience with large-scale data analysis [27]. The course has been taught at Wagner College for the last 6 years and has been highly successful in building an awareness of computational approaches in the sciences.

2 METHOD

Students used their smartphones to click pictures of various laboratory glassware routinely used in a Chemistry laboratory. They uploaded the pictures into a shared Google folder directly from their phones. This procedure was adopted to simplify the data collection process. Most of the pictures were taken with the goal of having one main object in the image. A mixture of empty and filled glassware was used to mimic a typical Chemistry laboratory setting. For example, beakers of various volume capacities were used: 250 mL, 500 mL, etc. The chemical composition of the solutions was not important for this exercise. Our intention was to introduce colors into the beakers to increase the sample space of pictures. A collection of sample images from each category is shown in Figure 3. Table 1 displays the number of classes and the number of images in each class in the dataset. A recent publication by Eppel et al. [3] implemented crowdsourcing to collect pictures of glassware in a chemistry laboratory. Their report is of much larger scope, with identification of the phase of the substance present inside the glassware. Our end-to-end exercise is designed with the express purpose of acquainting undergraduate students with the entire process, from collection and organization of raw images to analysis of final results.

The overarching idea was to collect pictures of glassware in a typical laboratory setting. Toward this end, some variability was also introduced in each class by intentionally including some background clutter. However, we realize that this may not be a best practice in terms of a practical goal of achieving the highest possible accuracy or success metric in object identification. Our goal was to get students to think through some of these issues during the collection of pictures. For example, the test tube collection class has pictures of multiple test tubes organized in a test tube stand. In this case, some test tubes were left empty, while others were partially filled with some of the prepared solutions. The test tube stand class has pictures of empty test tube stands of different types as well as partially-filled and fully-filled stands. Clearly, this is a nebulous area of labeling in our problem. However, that is a question of semantics, and our interest in this exercise was to demonstrate identification between our assigned labels. Some glassware is routinely seen suspended: for example, burets, separatory funnels, etc. In all such cases, we collected pictures of the glassware by placing them on a laboratory bench and also with their stands or supporting structures.

Figure 1 shows the collection of pictures of Erlenmeyer flasks that were used in the exercise. Some of the flask pictures were taken with empty flasks or with water in the flask. As mentioned earlier, colored solutions were also used in some of the pictures. A concerted effort was made to ensure that the pictures covered different volumes and with some variations in the contents of the flasks. The location of the flasks was also varied, and some pictures were taken on the

Table 1: Classes and number of images in each class

Class	Number of images
Beaker	30
Buchner funnel	20
Buret	11
Buret stand	5
Erlenmeyer flask	24
Flat bottom flask	18
Funnel	23
Graduated cylinder	47
Pipet	16
Round bottom flask	24
Separatory funnel	22
Standard measuring flask	36
Test tube	6
Test tube collection	17
Test tube stand	14
Viscometer	14
Wash bottle	8
Total	335

Figure 1: Erlenmeyer flask images collected by the students. Different colored solutions were used to fill up the flasks to various capacities.

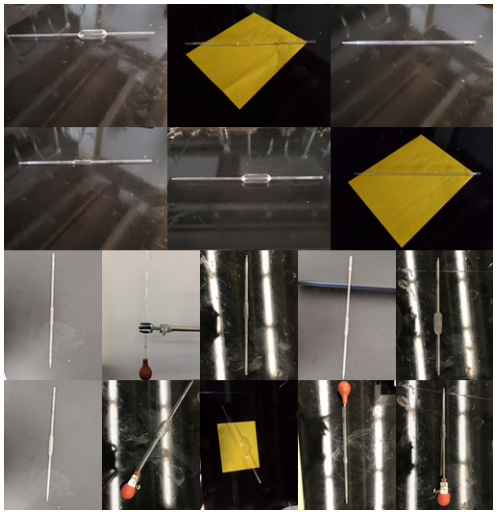


Figure 2: Images of pipets of various volume capacities in the dataset. Pipets seem to be difficult to differentiate from the background, and some images used a piece of paper to highlight the object.

laboratory bench, while others involved a common surface that was used for many pictures. Students used their own phones to collect pictures, and consequently there is considerable variation in the brightness, clarity, and contrast between the pictures. Some clutter, like faucets or electrical sockets, is visible in some of the pictures. Figure 2 displays the collection of pipet pictures that were part of the dataset. Pipets were particularly difficult to distinguish from surroundings under the lighting conditions in the laboratory, and some pictures utilized a small piece of colored paper to provide a suitable contrast for the pipet. Some pictures included a rubber bulb attached to a pipet. Additionally, it was quite difficult to get a clear picture of the 5 mL pipets, and some images incorporated a small piece of paper for easier differentiation. An additional picture was added with a pipet suspended from a stand to get a vertical orientation against a neutral wall background. In a similar fashion, buret pictures were also taken with a stand in the picture frame. Similar considerations were applied to all of the images in the dataset.

The image classification task was carried out with four neural networks that have demonstrated excellent results with the ImageNet competition data [8]. This allowed comparative studies and group-based investigations. Inception v1 [13] and Inception v3 [14] released by Google and ResNet-50 [19] and ResNet-101 [18] released by Microsoft were implemented in our exercise. All of these networks were trained on the ImageNet Large Scale Visualization Challenge 2012 classification dataset [23] consisting of 1.2 million images with 1,000 classes of objects. The plug-and-play nature of the pre-trained neural networks was also emphasized by implementing multiple neural networks. These networks are quite recent and well-known in image classification tasks. A brief overview of these networks is provided in Table 2.

Table 2: Four neural networks used for the image classification task. The pre-trained networks were downloaded from the Wolfram Neural Net Repository.

Network	Year	Source	Layers	Parameters
Inception v1	2014	Google	147	6,998,552
Inception v3	2015	Google	311	23,885,392
ResNet-50	2015	Microsoft	177	25,610,216
ResNet-101	2015	Microsoft	347	44,654,504

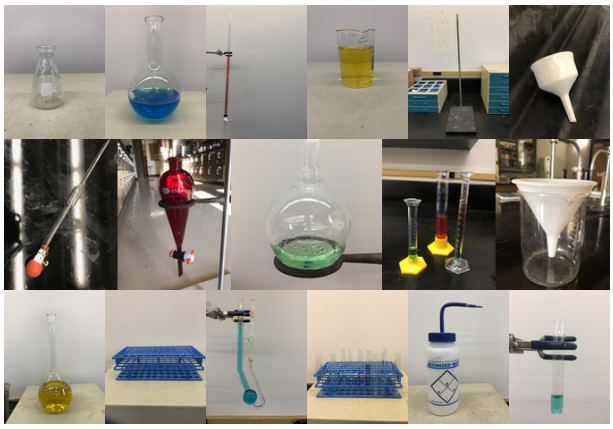


Figure 3: A sample of thumbnail-sized pictures from each of the classes in the dataset used for the classification process.

These pre-trained neural networks were downloaded from the Wolfram Neural Net Repository [22] and set up according to the instructions provided on the Wolfram website [21]. The training was performed by removing the final classification layers and replacing them with a classifier corresponding to the number of classes, 17, and a SoftMax layer to compute probabilities. The function NetDrop was used to perform these operations, and the training was performed using NetTrain. The training was carried out on a system with dual consumer class Graphical Processing Units for a maximum of 10 training rounds. The training performance is shown in Figure 5. The collected images were labeled, and the dataset was split into training and testing sets. 80% of the images were used for training, and the other 20% were reserved for testing. Since the population of items in the dataset is not uniformly distributed, the splitting of data into training and testing sets was carried out at the level of each class. This ensured that the training and testing sets contained each item of laboratory equipment. The training rounds with augmented images, and thus much larger number of samples, drops down in error during training much more rapidly as compared to the dataset with no augmentation of image samples. In either case, 10 training rounds seem to be sufficient in achieving a very low error during the training phase of the neural networks.

The image classification task performs best with small images, so the first step was to take the thumbnail version of all the images in the dataset. The following four datasets were constructed from the collected pictures to carry out this activity:

(1) Full color images captured by students
(2) Enhancement of the full color image dataset by image augmentation methods
(3) Grayscale images from the full color images
(4) Enhancement of the grayscale images by image augmentation methods

The step of image augmentation can be carried out through a hidden layer in the neural network. However, we chose to explicitly perform image augmentation to lead students to think through the steps of modifying images to enhance the dataset. The following module was used to carry out the image augmentation.

```
imageSetAugmentation[objectImages_List]:=Module[
{detailEnhanced,blurredImages,noisyImages,
lightDarkImages,reflectedImages,
rotatedImages,augmentedImages},
detailEnhanced=ImageEffect[#,"DetailEnhancing"]
&/@objectImages;
blurredImages=Blur[#,RandomInteger[{1,3}]]&
/@objectImages;
noisyImages=ImageEffect[#,"Noise"]&
/@objectImages;
lightDarkImages=Join[Lighter[#]&/@objectImages,
Darker[#]&/@objectImages];
reflectedImages=ImageReflect[#,Left]&
/@Join[objectImages,detailEnhanced,blurredImages,
noisyImages,lightDarkImages];
rotatedImages=ImageRotate[#,RandomInteger[{-10,10}]Degree]
&/@Join[objectImages,detailEnhanced,
blurredImages,noisyImages,lightDarkImages,
reflectedImages];
augmentedImages=Join[objectImages,
reflectedImages,rotatedImages];
Return[augmentedImages];
]
```

The Wolfram Language function ImageEffect was used to carry out detail enhancing and adding random noise effects to each image. Images were blurred using the Blur function with a randomly chosen pixel radius over which the blur was to be applied. Images were modified to appear lighter or darker using the appropriately named functions. Next, all of these images were collected and reflected from left to right. The final operation was to rotate all of these images with a randomly chosen rotation amount between -10 to 10 degrees. The result of all of these operations on one image taken from the set of Erlenmeyer flask images is shown in Figure 4. For every image in the raw dataset, 19 images were produced by the image augmentation procedure described above. The number of raw images in the dataset was 335, and after the implementation of the imageSetAugmentation module, the number of images increased to 6,365 for the two cases where image augmentation was applied. Thus, each network was trained and tested on 4 versions of the images. The versions without augmentation had 335 images in their complete dataset and the versions with augmentations had 6,365 images in their dataset.

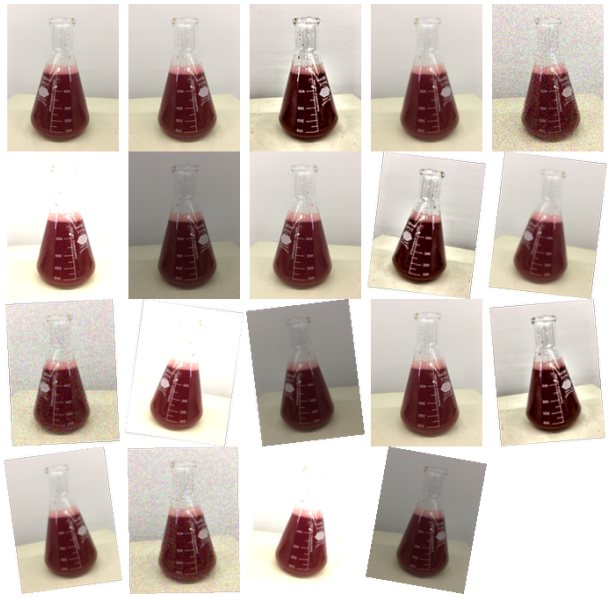


Figure 4: Image augmentation effects shown for an Erlenmeyer flask image. The images are subjected to blurring, rotation, reflection and changes in contrast as described in the text.

3 RESULTS AND DISCUSSIONS

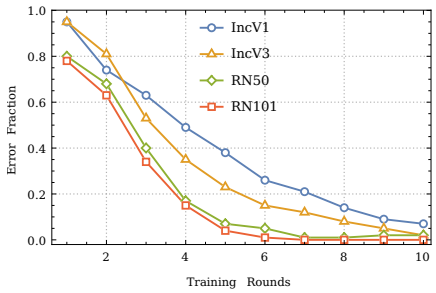
The training of each network resulted in a classifier trained to distinguish between the classes of the laboratory glassware in our training sample. These classifiers were then tested on the testing set generated for each set of images. The classification experiment for image sets without augmentation was carried out five times each, and the sets with augmentation were carried out three times each. The results of the classification performance on the testing set were compared using multiple metrics and are presented below.

3.1 Accuracy

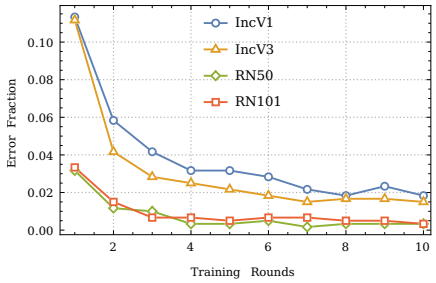
Accuracy is the fraction of correctly identified and labeled images from the testing set. The accuracy of classification is calculated as

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Examples}} \quad (1)$$

A graphical summary of the mean accuracy with standard error for the four networks and the four types of image datasets is shown in Figure 6. The plots show that there is essentially no difference in the training times for color images and grayscale images. The ResNet-50 network seems to provide a suitable trade-off between accuracy and training time in both cases, with and without image augmentation. There is a marked increase in accuracy with the application of image augmentation to increase the sample size for training. The highest classification accuracy of around 92% is lower than the least accuracy recorded, around 97% for the dataset enhanced with image augmentation methods. The image augmentation module increased the dataset size by a factor of 19, and a corresponding increase in training times can be seen from the plots. However, accuracy is not

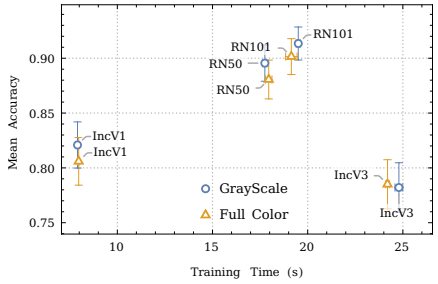


(a) Validation error during training for the set of images of glassware.

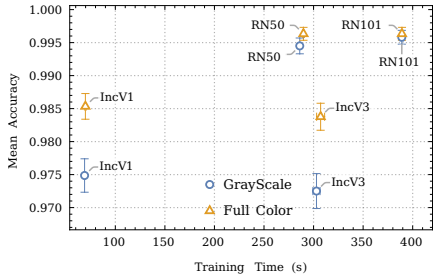


(b) Validation error during training for the set of images augmented with image modifications of glassware.

Figure 5: Validation error during training of all four neural networks. The error is negligible within ten training rounds. The validation error decreases significantly with the larger dataset of augmented images.



(a) Accuracy of classification and training times for the set of images of glassware.



(b) Accuracy of classification and training times for the set of images augmented with image modifications of glassware.

Figure 6: Accuracy of all four neural networks for each dataset. The ResNet-50 neural network provides the best trade-off between accuracy and training time. The datasets with image augmentation lead to much higher accuracy in classification.

a very reliable metric for a class-imbalanced dataset as present in this exercise.

We also analyzed the accuracy and rejection rate of samples based on different indeterminate threshold values. The maximum rejection rate is seen at an indeterminate threshold of around 90%. A more reasonable value of indeterminate threshold around 30% or 0.3 leads to accuracy around 99% . Figure 7 provides a graphical summary of results from the dataset of colored images with augmentation effects for the ResNet-50 network.

3.2 F_1 Score

The F_1 score is the harmonic mean of the precision and recall for the classification task. A high score implies that the classification produces a low number of false positives and false negatives. The values reported in Figure 8 are averages of the microaveraged F_1 score from each of the iterations. The microaverage F_1 score was calculated for each iteration to account for the differences in class frequencies. It is clear from Figure 8 that training on the augmented dataset enlarged with image effects gives rise to the highest values of F_1 scores for each case, full color images and grayscale images. The difference between ResNet performance and Inception performance is larger when the dataset is small. The calculation of the F_1 score is carried out as follows.

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2}$$

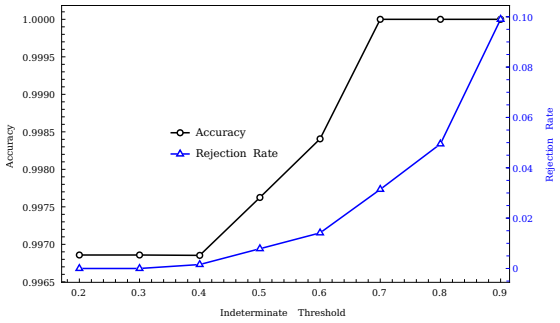
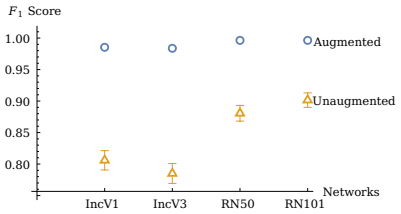


Figure 7: Accuracy and Rejection rate as functions of the threshold for indeterminate classification. Very high accuracy is observed for a wide range of classification thresholds for the dataset with augmented images.

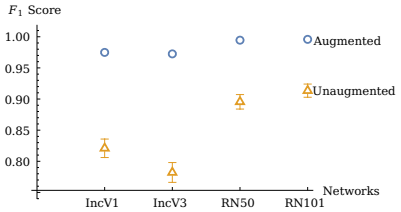
However, since this a multi-class problem, we computed the micro-averaged F_1 score. The micro-F_1 score is calculated as:

$$\text{Micro } F_1 \text{ score} = 2 \times \frac{\text{Micro precision} \times \text{Micro recall}}{\text{Micro precision} + \text{Micro recall}} \tag{3}$$

The calculation of the micro precision and micro recall are carried out as shown below. The acronyms have their usual meaning, TP



(a) F_1 score for full color images with and without augmentation.



(b) F_1 score for grayscale images with and without augmentation.

Figure 8: Microaveraged F_1 score for all datasets and neural networks in this exercise. The classification on augmented datasets gives rise to high F_1 scores in each case.

stands for true positives, FP is for false positives, and FN represents false negatives. The sums end at 17, because that is the number of classes in this classification problem.

$$\text{Micro precision} = \frac{TP_1 + \cdots + TP_{17}}{TP_1 + \cdots + TP_{17} + FP_1 + \cdots + FP_{17}} \quad (4)$$

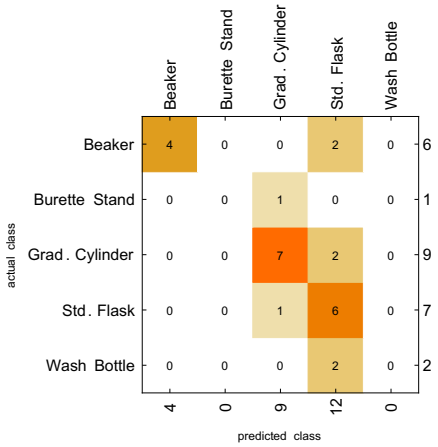
$$\text{Micro recall} = \frac{TP_1 + \cdots + TP_{17}}{TP_1 + \cdots + TP_{17} + FN_1 + \cdots + FN_{17}} \quad (5)$$

3.3 Confusion Matrix

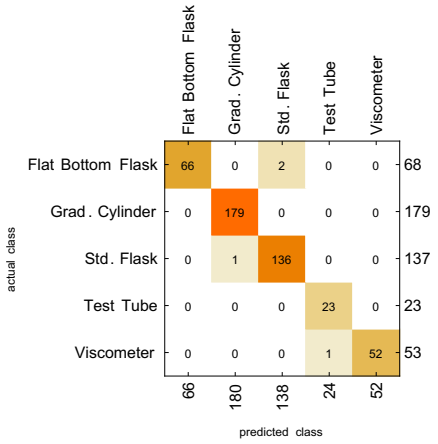
The confusion matrix is a succinct graphical representation of the confusions in the classes encountered by the classifier. Since the performance of ResNet-50 seems to be the most optimal, we highlight the confusion matrix for the top five confusions of this network for the case of augmented and unaugmented full color images. Figure 9 shows the confusion matrix, and it is evident that some of the confusions can be rationalized on the basis that the items in those classes indeed look quite similar to the human eye. For instance, a graduated cylinder is confused with a standard measuring flask, and a beaker is misclassified as a standard measuring flask. Such confusions, on a much smaller scale, also persist in the case of the dataset with augmented images. Another interesting example is that of a viscometer misclassified as a test tube. However, it is important to note that it is one misclassification out of 53 such images tested.

3.4 Geometric Mean Probability

Finally, the average and standard error of the geometric mean probability for the trials are shown in Figure 10. The geometric mean of the class probabilities provides an insight into the overall classification performance. Larger values of the geometric mean signify



(a) Confusion matrix for top five confusions for the set of unaugmented full color images.



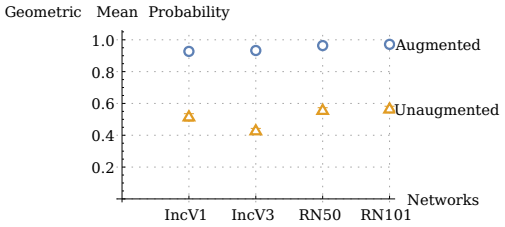
(b) Confusion matrix for top five confusions for the set of augmented full color images.

Figure 9: Confusion matrix plots for the classifier from the ResNet-50 network. The numbers on the bottom of each frame represent the number of correctly identified images, and numbers on the right edge of the frame are the total number of images for that class.

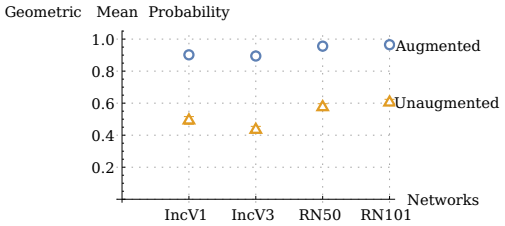
uniformly high confidence in the probabilities reported by the classifier during the testing phase. Figure 10 highlights the importance of augmentation and the resulting larger dataset for each case. The geometric mean probability appears insensitive to the color spectrum of the images and increases to values approaching 0.9 – 1.0 with the augmented datasets.

4 TEACHING IDEAS

The images of the dataset and sample notebooks used for training of networks and data analysis are freely available as Supporting Information. Short student projects to investigate the performance of classification for smaller number of classes may be constructed using the dataset. Students could be tasked with specific classes



(a) Geometric mean probability for the set of full color images.



(b) Geometric mean probability for the set of grayscale images.

Figure 10: Geometric mean of probabilities of actual class predictions. Part (a) is the set of full color images with and without augmentation, and part (b) is the set of grayscale images with and without augmentation. The augmented datasets in each case display a much larger value of the geometric mean, indicating stronger performance across the different classes.

of glassware images and asked to compare accuracy of classification. Another extension that could be implemented is to carry out image augmentation using different types of transformations and/or subsets of transformations from those that have been used in our implementation. The students could then investigate the effectiveness of those transformations towards the final classification performance. The exercise can also be extended by adding more glassware images and investigating classification performance. An interesting and possibly more advanced application would be to identify the text on glassware that annotates the volume, especially on beakers or Erlenmeyer flasks. Another application would be to identify the piece of glassware and to identify the hand-written or printed chemical species from the label attached to the glassware. This would integrate image classification and hand-writing recognition. The training of neural networks on the dataset with image enhancement is best carried out on systems with a GPU. The training times shown in this manuscript are result from execution on a dual-GPU workstation. However, the smaller datasets without image augmentation can be easily processed on workstations or laptops without a dedicated GPU. We imagine that instructors with limited resources could choose to carry out the training of augmented datasets on a dedicated workstation with a GPU, and students would work with the unaugmented datasets on their personal computing devices.

5 STUDENT FEEDBACK

This exercise was carried out with a cohort of eight first-year students in the author's Introduction to Scientific Computing course. The students expressed enthusiasm and interest toward more applications of machine learning following this exercise. Although there were no formal surveys, through informal feedback and one-on-one interviews, students pointed out that they enjoyed the project. They specifically enjoyed bringing their computing knowledge into the wet laboratory. Students with interest in biological sciences started discussions on applications of machine-learning methods to images obtained from microscopes. A majority of comments indicated that the activity helped them feel less intimidated about approaching machine learning or artificial intelligence related literature. They also reported increased interested in exploring computation as a tool toward their scientific domains of interest. A significant outcome of the informal feedback process was the realization from students that machine learning and advanced approaches are not limited to computer science majors or large technology companies.

6 CONCLUSIONS

We developed and implemented an end-to-end data science exercise with an application of machine learning experience for STEM students using their laboratory surroundings and equipment as the source of the project. Classification of images based on supervised learning is a common example in the machine learning domain, and the students adapted that into the chemistry laboratory. First-year students collected pictures of various glassware in the chemistry laboratory and implemented the training and testing of classifiers based on four pre-trained neural networks. These neural networks were chosen due to their wide availability and well-known performance on image classification tasks. The glassware images were split into two categories of full color images and grayscale images. Each set of images was enlarged with an image augmentation routine that resulted in a 19-fold increase in the size of the dataset. The students then compared the performance of the classification of glassware among the four networks and for each of the four types of datasets. The performance of the classifiers on the augmented datasets seems to be the most reliable, irrespective of using color images or grayscale images. Our analysis shows that ResNet-50 provides the best trade-off between accuracy and training time for the datasets considered in this activity. We believe that this activity provides students with an accessible and empowering introduction to advanced techniques in the data science domain through the lens of typical glassware in a chemistry laboratory.

7 SUPPORTING INFORMATION

We have provided the dataset of images and some of the Mathematica notebooks used to train the neural networks and to analyze the performance of the classifiers. The components are:

(1) Chemistry-Glassware-ML-no-augmentation-run1.nb: This notebook provides the code for setup and training of all four neural networks mentioned in the Methods for the dataset of full color images without image augmentation.
(2) Chemistry-Glassware-ML-with-augmentation-run1.nb: This notebook provides code for setup and training of the aforementioned neural networks for the dataset of full color images augmented with image modification effects.
(3) A folder called Glassware-Images contains images of the various glassware organized by name.

(4) Training and testing datasets for the first iteration of the experiment with no image augmentation with filenames training-Set-edison-2020-06-10T05:24:59.mx and testing-Set-edison-2020-06-10T05:24:59.mx
(5) Binary data export of neural networks trained on the laboratory glassware data. These files all have the .mx extension and the names start with trainedNet-*.mx. The name of the network is included in the filename string.

These resources are located in a shared Google drive folder. A copy of these resources is also hosted on Zenodo [25]. The dataset provides our trained networks with the extension ".mx," and the notebook entitled, "Analysis-run1-no-augmentation.nb" is set up with the correct filenames to load the trained networks and the testing and training dataset used for that iteration.

ACKNOWLEDGMENTS

We would like to record our appreciation and gratitude to students in the Introduction to Scientific Computing course. We would also like to thank Dr. Tuseeta Banerjee, Dr. Joshua Schrier, and Dr. Rishabh Jain for their helpful suggestions.

REFERENCES

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. 2016. End to End Learning for Self-Driving Cars. (apr 2016). arXiv:1604.07316 http://arxiv.org/abs/1604.07316
[2] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. 2016. An Analysis of Deep Neural Network Models for Practical Applications. arXiv:1605.07678 http://arxiv.org/abs/1605.07678
[3] Sagi Eppel, Haoping Xu, Mor Bismuth, Alan Aspuru-Guzik, and Cifar Lebovic Fellow. 2020. Computer vision for recognition of materials and vessels in chemistry lab settings and the Vector-LabPics dataset. (apr 2020). https://doi.org/10.26434/CHEMRXIV.11930004.V3
[4] Google. 2017. Explore the history of machine learning. https://cloud.withgoogle.com/build/data-analytics/explore-history-machine-learning/
[5] Google. 2020. Machine Learning Crash Course | Google Developers. https://developers.google.com/machine-learning/crash-course
[6] Thiago S. Guzella and Walmir M. Caminhas. 2009. A review of machine learning approaches to Spam filtering. , 10206–10222 pages. https://doi.org/10.1016/j.eswa.2009.02.037
[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *CoRR* abs/1502.01852 (2015). arXiv:1502.01852 http://arxiv.org/abs/1502.01852
[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Vol. 2016-Decem. IEEE, 770–778. https://doi.org/10.1109/CVPR.2016.90 arXiv:1512.03385
[9] Cheng Lung Huang, Mu Chen Chen, and Chieh Jen Wang. 2007. Credit scoring with a data mining approach based on support vector machines. *Expert Systems with Applications* 33, 4 (nov 2007), 847–856. https://doi.org/10.1016/j.eswa.2006.07.007
[10] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/
[11] Andrew Ng. 2020. Machine Learning by Stanford University | Coursera. https://www.coursera.org/learn/machine-learning
[12] Waseem Rawat and Zenghui Wang. 2017. Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Computation* 29, 9 (sep 2017), 2352–2449. https://doi.org/10.1162/neco_a_00990
[13] Wolfram Research. 2020. Inception V1 - Wolfram Neural Net Repository. https://resources.wolframcloud.com/NeuralNetRepository/resources/Inception-V1-Trained-on-ImageNet-Competition-Data
[14] Wolfram Research. 2020. Inception V3 - Wolfram Neural Net Repository. https://resources.wolframcloud.com/NeuralNetRepository/resources/Inception-V3-Trained-on-ImageNet-Competition-Data
[15] Wolfram Research. 2020. Machine Learning Basics Video Series: Wolfram U. https://www.wolfram.com/wolfram-u/machine-learning-basics/
[16] Wolfram Research. 2020. Machine Learning Courses and Classes: Wolfram U. https://www.wolfram.com/wolfram-u/catalog/machine-learning/
[17] Wolfram Research. 2020. Overview of Machine Learning in the Wolfram Language: Wolfram U Class. https://www.wolfram.com/wolfram-u/catalog/wl030/
[18] Wolfram Research. 2020. ResNet-101 - Wolfram Neural Net Repository. https://resources.wolframcloud.com/NeuralNetRepository/resources/ResNet-101-Trained-on-ImageNet-Competition-Data
[19] Wolfram Research. 2020. ResNet-50 - Wolfram Neural Net Repository. Retrieved June 22, 2020 from https://resources.wolframcloud.com/NeuralNetRepository/resources/ResNet-50-Trained-on-ImageNet-Competition-Data
[20] Wolfram Research. 2020. Supervised Machine Learning: Input & Output: Wolfram U Class. https://www.wolfram.com/wolfram-u/catalog/wl031/
[21] Wolfram Research. 2020. Train a Custom Image Classifier: New in Wolfram Language 12. https://www.wolfram.com/language/12/machine-learning-for-images/train-a-custom-image-classifier.html?product=mathematica
[22] Wolfram Research. 2020. Wolfram Neural Net Repository of Neural Network Models. https://resources.wolframcloud.com/NeuralNetRepository/
[23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* 115, 3 (dec 2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y arXiv:1409.0575
[24] Jonathan Schmidt, Mário R. G. Marques, Silvana Botti, and Miguel A. L. Marques. 2019. Recent advances and applications of machine learning in solid-state materials science. *npj Computational Materials* 5, 1 (dec 2019), 83. https://doi.org/10.1038/s41524-019-0221-0
[25] Arun Sharma. 2020. Glassware images and code samples for training and identification of glassware by neural networks. https://doi.org/10.5281/zenodo.4019356
[26] Arun K Sharma. 2017. A model Scientific Computing course for freshman students at liberal arts Colleges. *The Journal of Computational Science Education* 8, 2 (jul 2017), 2–9. https://doi.org/10.22369/issn.2153-4136/8/2/1
[27] Arun K Sharma, Michelle Hernandez, and Vinh Phuong. 2019. Engaging Students With Computing And Climate Change Through A Course In Scientific Computing. *Journal of STEM Education* 20, 2 (2019), 5–13. https://www.jstem.org/jstem/index.php/JSTEM/article/view/2409
[28] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (jan 2016), 484–489. https://doi.org/10.1038/nature16961

Transport Phenomena in High-speed Wall-bounded Flows Subject to Concave Surface Curvature

Ernie Rivera
NAVAIR Air Station
Patuxent River, MD

Guillermo Araya
University of Puerto Rico at Mayagüez
Mayagüez, PR
araya@mailaps.org

ABSTRACT

Turbulent boundary layers that evolve along the flow direction are ubiquitous. Moreover, accounting for the effects of wall-curvature driven pressure gradient and flow compressibility adds significant complexity to the problem. Consequently, hypersonic spatially-developing turbulent boundary layers (SDTBL) over curved walls are of crucial importance in aerospace applications, such as unmanned high-speed vehicles, scramjets, and advanced space aircraft. More importantly, hypersonic capabilities would provide faster responsiveness and longer range coverage to U.S. Air Force systems. Thus, the acquired understanding of the physics behind high speed boundary layers over curved wall-bounded flows can lead to the development of more efficient control techniques for the fluid flow (e.g., wave drag reduction) and aerodynamic heating on hypersonic vehicle design. In this investigation, a series of numerical experiments is performed to evaluate the effects of strong concave curvature and supersonic/hypersonic speeds (Mach numbers of 2.86 and 5, respectively) on the thermal transport phenomena that take place inside the boundary layer. The flow solver to be used is based on a RANS approach. Two different turbulence models are compared: the SST (Shear Stress Transport) model by Menter and the standard k-ω model by Wilcox. Furthermore, numerical results are validated by means of experimental data from the literature (Donovan *et al.*, J. Fluid Mech., 259, 1-24, 1994) for the moderate concave curvature case and a Mach number of 2.86. The present study allows us to initially obtain a first insight of the flow physics for a forthcoming better design of 3D meshes and computational boxes, as part of a more ambitious project that involves Direct Numerical Simulation (DNS) of curved wall-bounded flows in the supersonic/hypersonic regime. The uniqueness of this RANS analysis in concave curved walls can be summarized as follows: (i) study of the compressibility effects on the time-averaged velocity and temperature, (ii) analysis of the influence of different inflow boundary conditions.

KEYWORDS

RANS, turbulent boundary layer, supersonic, hypersonic, wall curvature

https://doi.org/10.22369/issn.2153-4136/12/1/3

1 INTRODUCTION

Significant research effort has been devoted to high speed flight in the last decades, since it is directly connected to "*rapid responsiveness, increased survivability in contested environments and efficient range coverage*" from a military perspective, according to Schmisseur [12]. What is more, a Mach 6-aircraft would be able to reach the US West Coast in approximately 23 minutes from the US East Coast [12]. In 2013, the Boeing X-51A Waverider Scramjet prototype was released at 50,000 feet as part of the the fourth and final test flight in the U.S. Air Force program, reaching a Mach number of 5.1, which is enough to fly from New York to London in roughly 75 minutes.

Furthermore, due to the complex geometries associated with these high-speed aircraft (as seen in Figure 1); such as unmanned hypersonic vehicles (e.g., Boeing X-51 WaveRider and NASA X-43), scramjets, and space planes; surface curvature plays a crucial role in the boundary layer physics and aerothermodynamics. This is attributed to the combined effect of pressure gradients and streamline curvature (Spina *et al.* [13]), which induces extra strain rates to the main shear ($\partial U/\partial y$) associated with streamline curvature ($\partial V/\partial x$), pressure gradients ($\partial p/\partial x$ and $\partial p/\partial y$), and bulk compression or dilatation ($\nabla \cdot \mathbf{V}$) (Donovan *et al.* [5]). The principal features of curved wall-bounded flows (so-called Görtler flows) are the presence of a centrifugal force and a pressure gradient (i.e., $\partial p/\partial n$) acting on the fluid in the wall-normal direction as well as a streamline pressure gradient (i.e., $\partial p/\partial s$) in curvilinear coordinates (s, n).

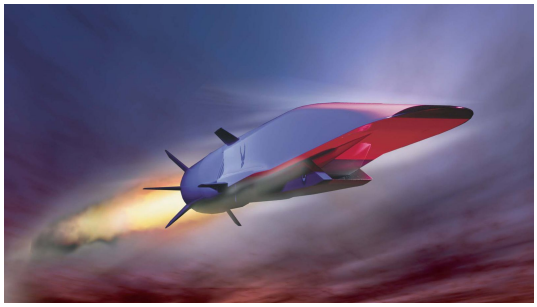


Figure 1: Cartoon of the Boeing X-51A Waverider (source: Wikipedia)

An exhaustive review of pressure gradient and streamline surface curvature effects on the behavior of supersonic turbulent boundary layers can be found in Spina *et al.* [13]. This revision was focused on experimental studies and two-dimensional geometries, but not on the distortion of compressible turbulent boundary layers by shock

waves. Furthermore, concave curvature induces a destabilizing effect on the flow by enhancing turbulent mixing, whereas convex curvature is stabilizing. Thomann [17] experimentally studied the isolated effects of wall curvature surfaces (concave and convex) on the heat transfer (Stanton number) in Mach-2.5 turbulent boundary layers by eliminating the streamwise pressure gradients. He found an increase of 20% on Stanton numbers along the concave wall, while an analogous decrease was observed for the convex wall. Historically, the combined effect of concave surface and Adverse Pressure Gradient (APG) on supersonic/hypersonic turbulent boundary layers has been the motivation of several experimental studies in curved 2D ramps and flared cones, for instance [9] [6] [14] [15] [8]. Smits and his colleagues at Princeton University have performed a series of experiments in two-dimensional concave walls with the objective of gaining insight into the effects of different radii of curvature and turning angles on adverse pressure gradient strength and bulk compression [16] [7] [5]. Furthermore, the typical manifestation of destabilizing effects in concave surfaces in subsonic flow is the generation of Taylor-Görtler-type (T-G) vortices, even in turbulent flows [3] [13]. Jayaram *et al.* [7] observed significant increases of turbulence levels, structural parameters (such as the stress ratio), and length/time scales of turbulent motions in the larger-curvature case. Donovan *et al.* [5] found a significant amplification of the Reynolds stresses, and the streamwise length of the average large-scale motions approximately doubled in concave surfaces at a Mach number of 2.86. As concluded in [7] and [5], concave curvature provokes an increase of the wake strength and a dip below the log law in the mean streamwise velocity as well as an emergence of an outer secondary peak on the streamwise component of the Reynolds normal stresses ($\overline{u'^2}$). These features are attributed to the presence of a streamwise APG on the flow provoked by the concave surface. Similar peculiarities were reported by Araya *et al.*[2] and Araya and Castillo [1] in incompressible turbulent boundary layers subject to moderate and strong streamwise APG on flat surfaces.

In summary, the performed literature review has revealed the strong influence of wall-curvature driven pressure gradients on the mean flow and heat transfer inside a compressible SDTBL. It is clear that the accurate and comprehensive knowledge of the curved wall effects on the physics of supersonic/hypersonic SDTBL will lead to the development of flow control mechanisms on high speed vehicle design. This article focuses on the study of supersonic/hypersonic SDTBL under the influence of strong streamline concave curvature, which can lead to the determination of appropriate flow control tools and design optimization in high speed science.

2 MESH GENERATION, FLOW SOLVER, AND BOUNDARY CONDITIONS

Figure 2 (a) shows a schematic of the computational domain for the strong concave curvature, or Case 1 in Table 1. The proposed geometry in Case 1 is based on the experimental study of Donovan *et al.* [5] and covers full spatial dimensions of the experimental model. The available experimental results, such as wall pressure and wall shear stress, are used for our RANS validation. Furthermore, the experimental boundary layer in [5] evolves along the streamwise direction; however, it shows a homogeneous spanwise condition,

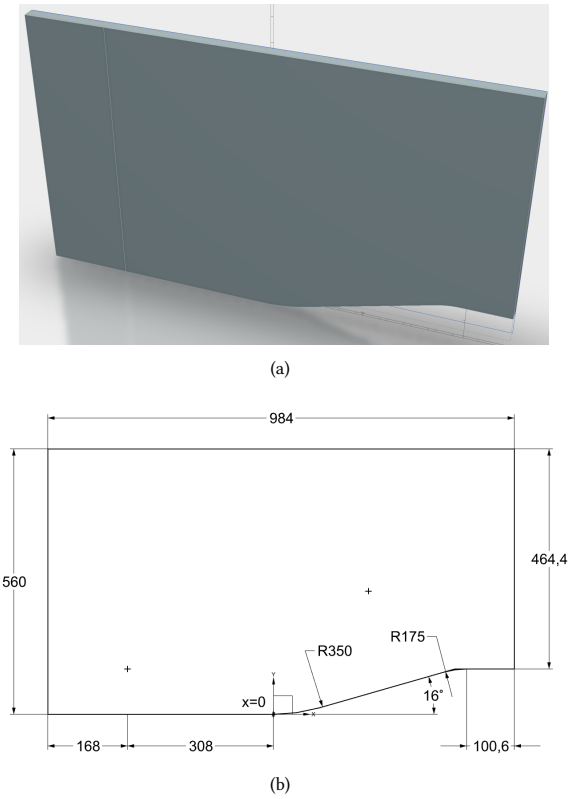


Figure 2: (a) Schematic of the computational domain, and (b) domain dimensions in mm.

and consequently, the mean flow is two-dimensional. The wall curvature is prescribed based on the radius of curvature, R, defined in Table 1 in terms of the reference boundary layer thickness, δ_{ref}. Consistent with experiment, the value of δ_{ref} (= 28mm) is taken at the origin of the coordinate system (x = 0 mm or beginning of the curved surface). Here, x is the streamwise distance along the model surface, and y is the wall-normal coordinate. The curved wall ends at x = 98 mm, or approximately $3.5\delta_{ref}$. The wall-curvature driven pressure gradient zone induces an Adverse Pressure Gradient (APG) on the flow. Upstream, there is a Zero-Pressure Gradient (ZPG) zone, which serves as a point of reference (i.e., baseline cases) to assess the effects of wall curvature on the flow. In Figure 2 (b), the corresponding 2D domain dimensions (in mm) for Case 1 can be observed.

Table 1 summarizes the characteristics of the proposed two (2) cases according to the wall curvature (δ_{ref}/R), Mach number (M_∞), momentum thickness-Reynolds numbers ($Re_\theta = \rho_\infty U_\infty \theta/\mu_\infty$, $Re_{\delta 2} = \rho_\infty U_\infty \theta/\mu_w$, based on the free-stream and wall viscosity, respectively), and computational domain dimensions in terms of δ_{ref} (where subscript ∞ stands for free-stream values, w stands for wall values, and L_x and L_y represent the streamwise and wall-normal domain lengths, respectively). Compressibility effects are taken into account by means of two different Mach numbers: 2.86 and 5, for cases 1 and 2, respectively, and by prescribing the same strong

surface curvature, i.e. δ_{ref}/R = 0.08 in both cases. These two values for Mach numbers indicate that selected cases are in the supersonic regime and in the lower limit of hypersonic flows.

Flow Solver: The STAR-CCM+ package for computational fluid dynamics is selected to solve the governing equations of compressible flow in this investigation. The following turbulence models are prescribed for each case in Table 1: the Shear Stress Transport (SST) model by Menter [10] and the standard $k-\omega$ model by Wilcox [18]. Two–equation turbulence models are complete, because transport equations are solved for both turbulent scales, i.e. the velocity and the length scale. The original $k-\omega$ model [18] exhibits a freestream dependency of ω, which is generally not present in the $k-\epsilon$ model. Menter [10] integrated the advantages of both models via blending functions, which permitted switching from $k-\omega$, close to a wall, to $k-\epsilon$, when approaching the edge of a boundary layer. A further improvement by Menter [10] was a modification to the eddy viscosity, based on the idea of the Johnson-King model, which established that the transport of the main turbulent shear stresses was crucial in the simulations of strong Adverse Pressure Gradient (APG) flows. This new approach was called the Menter shear–stress transport model (SST). In particular, the Menter SST turbulence model is well-known for its good performance in boundary layer flows subjected to APG or flow deceleration. Since the concave surface curvature induces a strong deceleration on the flow, one of the purposes of the present study is to evaluate the SST's performance on concave walls.

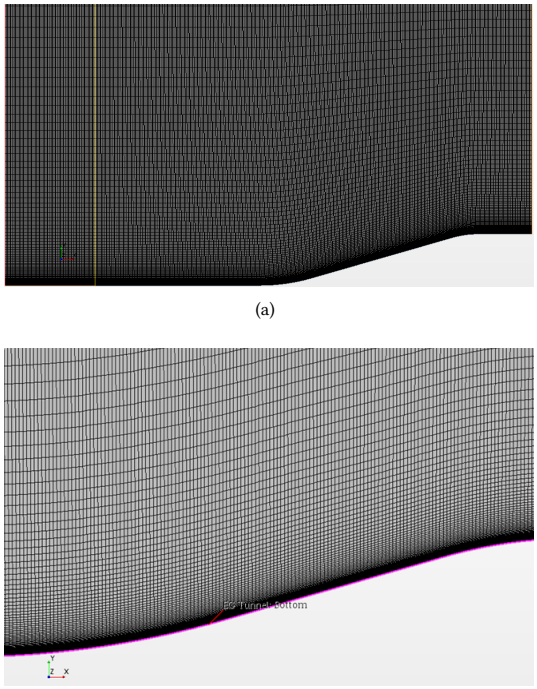
Boundary Conditions: At the wall, the classical no-slip condition is imposed for velocities. Isothermal wall is assumed for the thermal field with $T_w = 280.8K$ as in [5]. The working fluid is calorically perfect non-reacting air. At the inlet boundary, four different options are tested and compared, which are described in detail in Section 3. At the top surface, freesstream values are prescribed.

Table 1: Numerical cases with concave surface curvature.

Case	δ_{ref}/R	M_∞	$Re_\theta/Re_{\delta 2}$	$L_x \times L_y$
1	0.08	2.86	82,000 / 38,140	$35\delta_{ref} \times 20\delta_{ref}$
2	0.08	5	143,360 / 66,679	$35\delta_{ref} \times 20\delta_{ref}$

3 NUMERICAL RESULTS

For Case 1, the inlet free-stream velocity U_∞ is set to 581 m/s (M_∞ = 2.86), whereas U_∞ is set to 1014 m/s (M_∞ = 5) for Case 2. In all cases, the static free-stream temperature T_∞ is 102.4K. The grid is 723 (streamwise) by 200 (wall-normal) grid points. The mesh is stretched in the wall-normal direction with the first off-wall point located at 6×10^{-8} m. The first off-wall point in the Mach-5 case is placed at $\Delta y^+ \approx 0.09$, which ensures an appropriate near wall resolution ($\Delta y^+ < 1$). Figure 4 exhibits the time variation of flow residuals for continuity, momentum, energy, and turbulence transport (i.e., for turbulent kinetic energy, tke, and specific dissipation rate) equations. It is observed that numerical convergence is achieved in Case 1 by employing the SST turbulence model after 30,000 iterations with a CFL parameter of 0.5.



(a)

(b)

Figure 3: Computational mesh (a) total domain (flow from left to right) and (b) close-up of the curved surface.

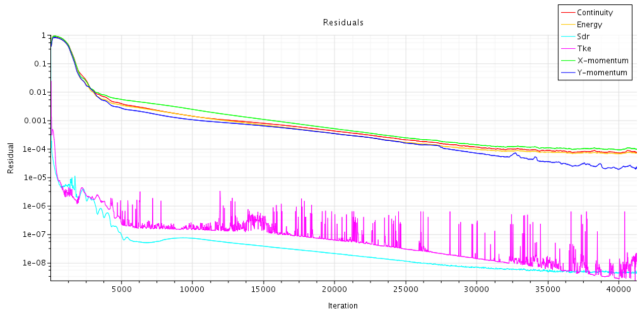


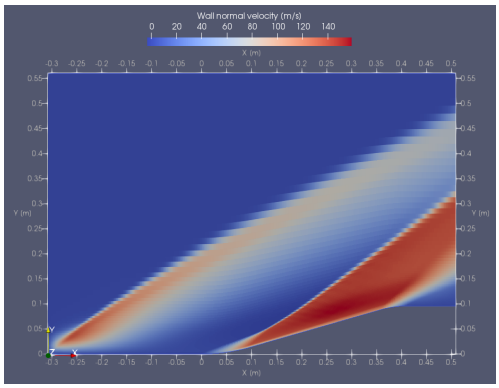
Figure 4: Time history of flow residuals.

Different methodologies for the inlet boundary condition are tested in Case 1. Figure 5 depicts iso-contours of the wall-normal velocity based on the SST turbulence model. In Figure 5 (a), the streamwise velocity U was assigned a 1/7 power law profile inside the boundary layer with a U-parabolic profile for the static temperature T. In Figure 5 (b) a composite velocity profile is prescribed that consists of Reichardt's [11] inner layer profile, Finley's wake function (Cebeci & Bradshaw [4]), and the Walz equation for the temperature profile. In Figure 5 (c) the velocity and thermal profiles were extracted from $x = -0.0625m$ in the previous case and re-injected at the domain inlet (this case is called "recycle"), whereas in 5 (d) a symmetry zone is attached upstream of the flat plate edge.

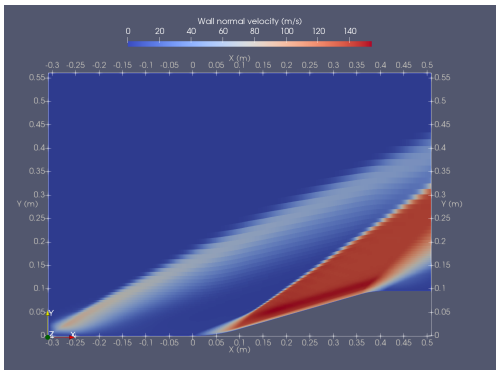
Clearly, an inclined line of disturbances is observed in Figure 5 (a) at approximately 20° with respect to the $x-$direction, very close to the Mach angle $\mu = sin^{-1}(1/M_\infty)$, which is generated at the edge of the flat surface (no-slip condition) due to improper inlet flow profiles. This Mach wave is somehow reduced by setting the Reichardt's and Walz's equations at the inlet (see Figure 5 (b)), but not completely. The recycling technique in Figure 5 (c) has properly modeled inflow conditions, since the inclined disturbances almost disappeared. In a similar way, by attaching an upstream symmetry condition (see Figure 5 (d)) and resolving turbulence transition, the sonic disturbances are minimized. However, in this case, the presence of the Mach wave is physical (not artificial) due to the interaction of supersonic free-stream with the edge of the flat plate. Therefore, the importance of setting realistic turbulent inflow conditions is unquestionable. Consequently, in the present study, an upstream slip boundary for modeling the inlet conditions is employed for the rest of the manuscript.

Figure 6 exhibits a comparison of present numerical results of Case 1 with the experimental wall static pressure. Generally speaking, both turbulence models (SST and $k-\omega$) capture quite well the significant increase (up to three times) of the upstream surface pressure due to the presence of the concave curvature (i.e. for $0m \leq x \leq 0.098m$). Beyond the end of the curved wall, more precisely in the inclined straight surface, some discrepancies with experimental values (in the order of 5%) can be observed. However, the agreement is very good between numerical results for the SST model and inlet recycled profiles with Donovan's experimental data, particularly beyond the concave curvature (i.e. for $x > 0.098m$). This might be caused by a higher incoming Reynolds number prescribed in the inflow recycled profile case.

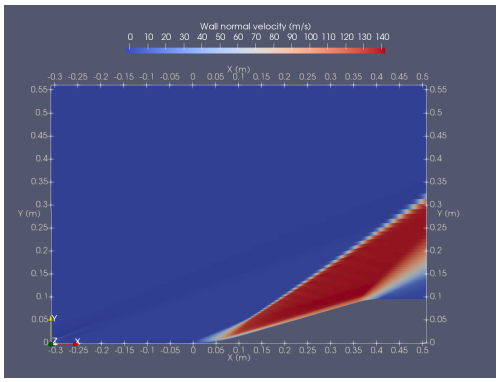
The skin friction coefficient ($C_{f,ref}$), defined as the wall shear stress normalized by the upstream free-stream density and velocity, is plotted in Figure 7. Experimental data from [5] is included. It can be seen that both turbulence models and inlet free-stream conditions exhibit similar performance. After the typical decreasing trend in the zero-pressure gradient region, numerical results of $C_{f,ref}$ based on the SST (free-stream inlet) and $k-\omega$ models significantly over-predict experimental values. On the other hand, the SST model (recycled profile inlet) shows an anomalous increasing trend of $C_{f,ref}$ in the ZPG zone; nevertheless, the obtained skin friction coefficient in the concave wall exhibits a much better agreement with experiments, which may be attributed to the higher Reynolds number imposed. Moreover, the observed increase in the skin friction coefficient is contrary to what occurs in incompressible boundary layers [2] subject to adverse pressure gradient, where the wall shear stress decreases. This is explained by the fact that the density increases more than the velocity decreases (flow deceleration) in compressible flows. Therefore, the boundary layer thickness based on the time-averaged streamwise velocity (not density-averaged) enlarges, as it will be shown later on, and the wall velocity gradient increases, as well. Figure 8 depicts the mean streamwise velocity at $x = 40$ mm (halfway through the curve). The SST and $k-\omega$ turbulence models (free-stream inlet) perform similarly with over-predictions of the order of 12% with respect to experiments from [5] at $y/\delta \approx 0.3$. On the other hand, the SST model (recycled profile inlet) exhibits a significant improvement when



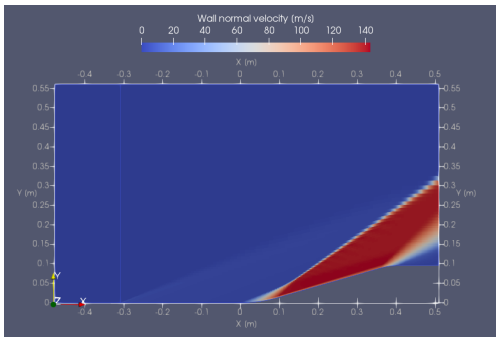
(a)



(b)



(c)



(d)

Figure 5: Iso-contours of wall-normal velocity for Case 1 and different inflow conditions.

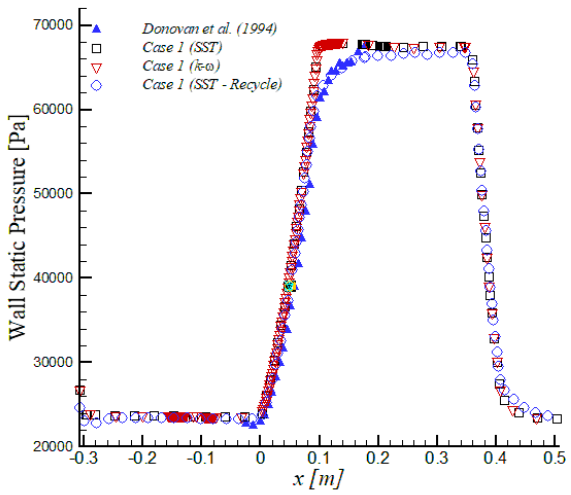


Figure 6: Wall static pressure.

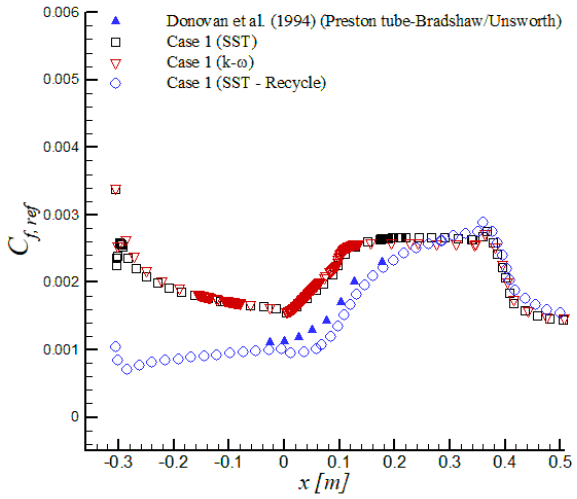


Figure 7: Skin friction coefficient.

the incoming Reynolds number is higher. A good agreement is observed with experimental data from [5], particularly for $y/\delta > 0.25$. The mean temperature and mean streamwise velocity at $x = 40$ mm (halfway through the curve) are plotted in Figure 9. In general, analogous trends of the analyzed three cases were observed when compared to the Walz's equation:

$$\frac{T}{T_\infty} = \frac{T_w}{T_\infty} + \frac{T_r - T_w}{T_\infty}\left(\frac{U}{U_\infty}\right) - r\frac{\gamma - 1}{2}M_\infty^2\left(\frac{U}{U_\infty}\right)^2 \quad (1)$$

where T_w is the wall temperature, T_r is the adiabatic or recovery temperature, r is the recovery factor (= $Pr^{1/3}$, where Pr is the Prandtl number), and γ is the specific heat ratio for air (= 1.4).

Figure 10 depicts the streamwise variation of the local boundary layer thickness for Case 1 by considering the SST model and inlet free-stream conditions vs. recycled profiles, respectively. The recycling method has generated much larger values of the inlet boundary layer thickness (i.e., $\delta \approx 20$ mm), very close to the experimental δ_{ref} of 28 mm as in [5]. On the contrary, the inlet free-stream condition has generated very small boundary layer thicknesses, indicating that a much longer ZPG zone would be necessary. It is inferred that, by means of this method, it is hard to control the desired inflow boundary layer thickness, requiring a significantly long inlet section to achieve the reference or target δ_{ref}, and consequently, penalizing computational resources. Due to the presence of APG in the concave wall curvature from $x = 0$ mm, the boundary layer significantly grows (up to 25%). Downstream of the curved wall, the flow recovers and accelerates in the inclined ramp at a turning angle of 16°, where nearly constant values in the wall static pressure (see this ZPG region around $0.15m < x < 0.35m$ in Figure 6) and in the boundary layer thickness are observed (see Figure 10). Beyond $x \approx$ 400 mm, the convex wall curvature strongly accelerates the flow causing a relaminarization process. Additionally, a shrinking trend is seen for the boundary layer thickness along the convex wall. In Figure 11, a similar behavior of the Re_θ streamwise variation can be seen. By the end of the concave curvature ($x \approx 98mm$), the Re_θ has increased around 6 times regarding the incoming value.

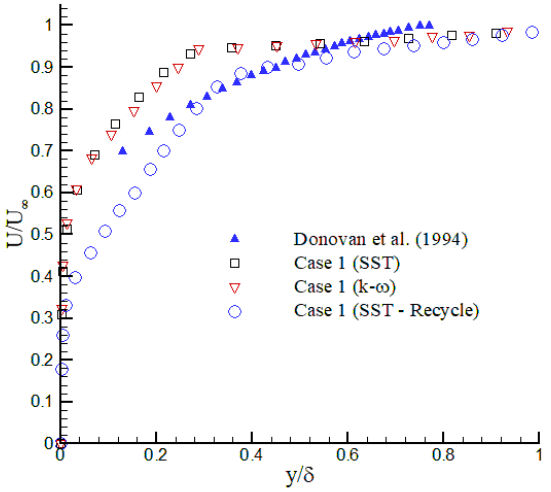


Figure 8: Mean streamwise velocity at $x = 40$ mm (halfway through the curve).

Figure 12 depicts iso-contours of streamwise velocity and Mach number for Case 1 and the SST turbulence model with inlet free-stream conditions. At the inlet zone, the presence of a Mach cone inclined at approximately 20° with respect to the streamwise direction due to flow perturbations (developing section) is nearly imperceptible. The lesson learned here is the importance of prescribing realistic turbulent inflow conditions. The strong concave wall curvature induces a significant deceleration on the flow or APG zone and formation of compression waves with decreasing values of the Mach numbers around 2. These compression waves formed a cone at approximately 35°, which are fully convected at the outlet plane without bouncing over the top surface. Therefore, future planned DNS studies on concave surfaces will have to consider at least $11\delta_{ref}$-tall computational domains. In spite of the

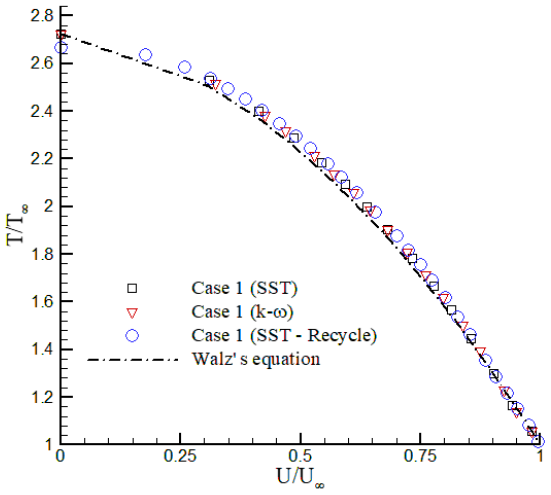


Figure 9: Mean temperature vs. mean streamwise velocity at x = 40 mm (halfway through the curve).

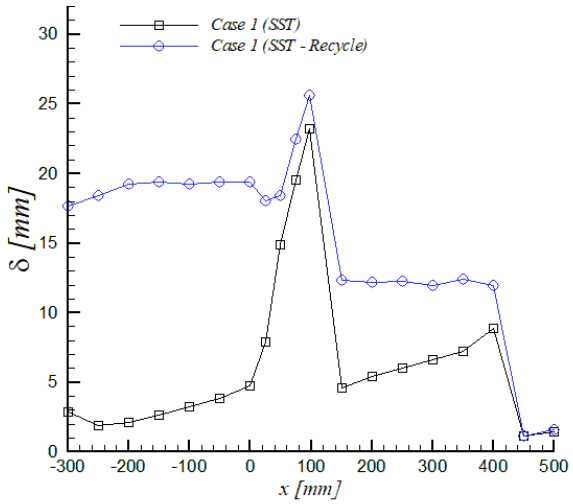


Figure 10: Streamwise variation of the boundary layer thickness δ.

presence of a strong APG region, the flow remains fully attached in the curved-inclined wall region, confirmed by positive values of $C_{f,ref}$ in Figure 7.

The static temperature is shown in Figure 13 (a). The flow experiences a thermal increase of roughly 70K across the compression waves. Similar to the velocity boundary layer, it seems that the thermal boundary layer goes through a thickening process along the curved surface with an increase of the wall heat flux. From Figure 13 (b), it can be inferred that a strong concave curvature (with a curvature radius twelve times larger than the boundary layer thickness) may induce a compression ratio in the order of 2.5 at $M_\infty \sim 3$. In Figure 14, the effects of compressibility on the wall static pressure can be observed. In Case 2, the Mach number was set to 5 (hypersonic regime), and the selected turbulence model was SST with inlet free-stream conditions. The hypersonic regime provokes an approximate increase of 100% in the wall static pressure over the curved surface. Iso-contours of the Mach number in Figure 15 (a) show a more tilted (with respect to the streamwise direction ($\sim 26°$)) and confined zone of compression waves at the hypersonic level. While the compression ratio is in the order of 6 for Case 2 (see Figure 15 (b)).

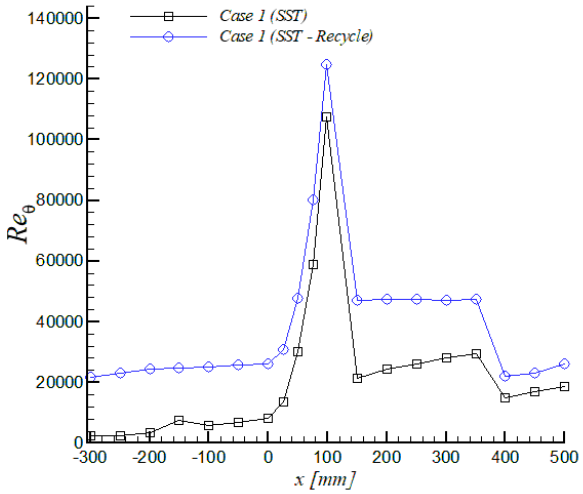
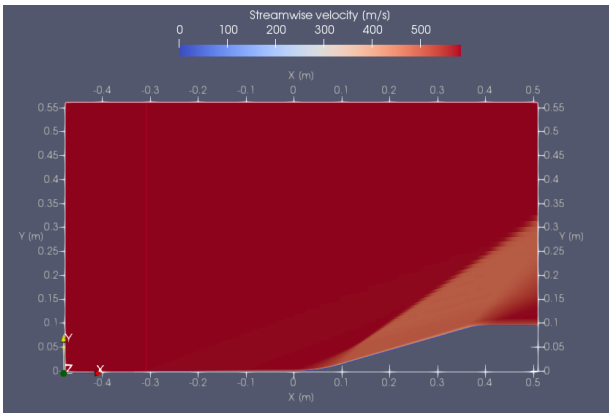


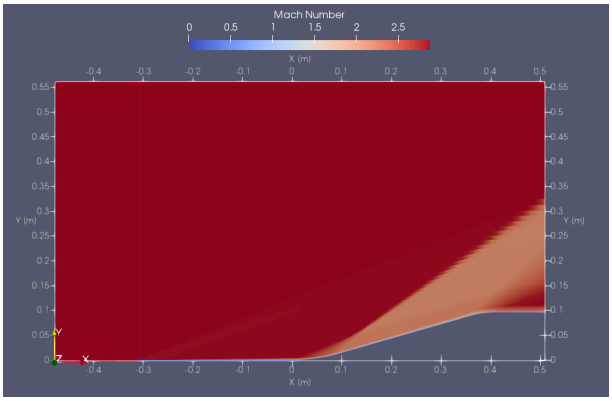
Figure 11: Streamwise variation of the Reynolds number Re_θ.

4 CONCLUSIONS

RANS simulations are performed in order to study the combined effects of wall concave curvature and compressibility. The computational domain is prescribed as in wind tunnel experiments by Donovan *et al.* [5]. In addition, a case is designed to shed some light on the influence of strong concave curvature and hypersonic speeds on the hydrodynamic/thermal field. The presence of strong curvature on spatially-developing turbulent boundary layers at Mach = 2.86 induces an Adverse Pressure Gradient (APG) with a subsequent increase of the wall-shear stresses and wall-heat fluxes. The SST and $k-\omega$ turbulence models with free-stream inlet conditions have demonstrated similar performance when compared to experiments. A clear supremacy of the SST over the standard $k-\omega$ model has not been identified in the APG region, at least for the present conditions. However, the SST model based on recycled inflow profiles has shown a good agreement with experimental data by [5], since the incoming Reynolds number is higher. It has been estimated that a separate simulation of a flat plate turbulent boundary layer would add an extra 50% in terms of computational resources in order to extract, recycle, and inject the required inflow conditions to the principal domain. Nevertheless, and based on the quality of the obtained results, the extra resources pay off. The compression waves formed a 35° cone; therefore, it is important to ensure enough room for the top surface in order to avoid wave reflections back to the domain. By increasing the inlet Mach number

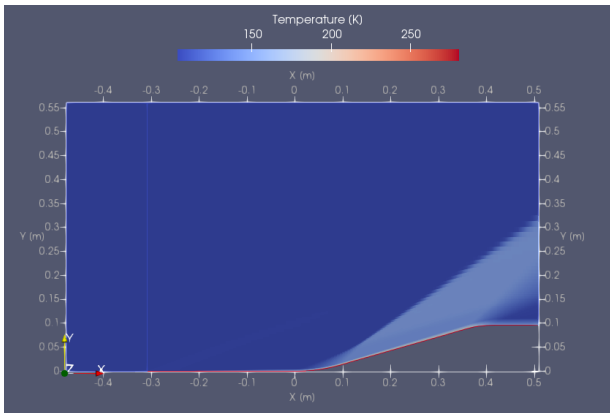


(a) Streamwise velocity

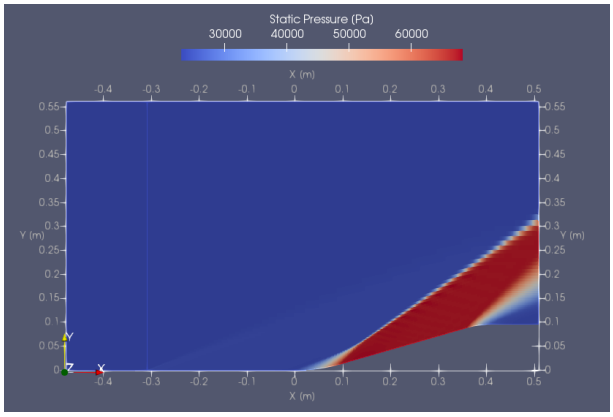


(b) Mach number

Figure 12: Iso-contours for Case 1 and SST turbulence model.



(a) Temperature



(b) Static pressure

Figure 13: Iso-contours for Case 1 and SST turbulence model.

at the hypersonic regime, a more restrained compression zone has been observed with a significant increase in the compression ratio.

5 REFLECTIONS

I had little experience working with Computational Fluid Dynamics (CFD) prior to the Blue Waters Student Internship Program. For this reason, the experience has been a learning challenge and has broadened my knowledge in fluid dynamics, specifically in the area of supersonic and hypersonic flows. There were many challenges surrounding the underlying theory of fluid dynamics, specifically concerning the boundary layer in supersonic and hypersonic flows. Also, learning how to identify the cause of issues present in the simulation required persistence and in some cases creativity to resolve such issues. These issues usually highlighted areas where more learning was required. The internship experience helped me to learn more about turbulence and CFD.

6 ACKNOWLEDGMENTS

This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-17-1-0051. ER acknowledges the Blue Waters Student Internship Program. This research is part of the Blue Waters sustained-petascale computing

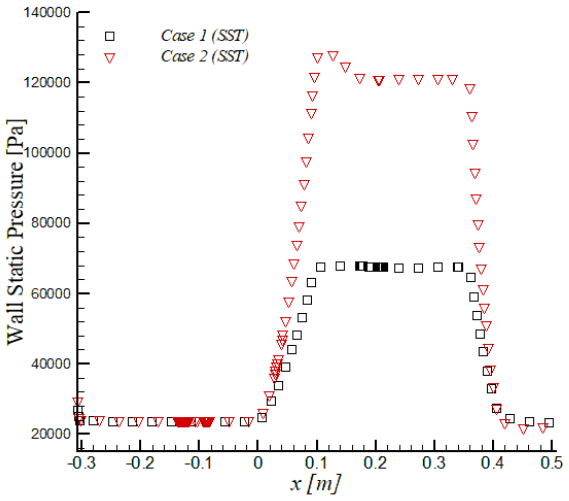
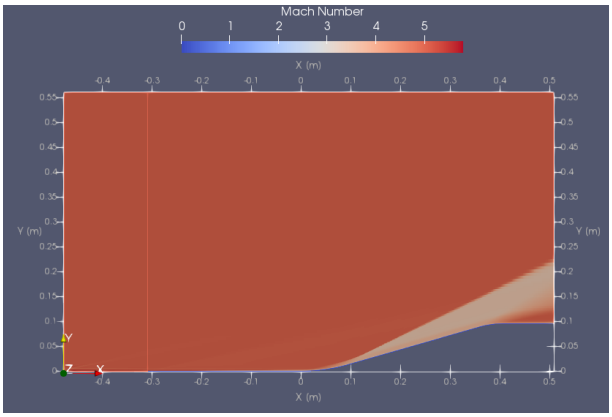
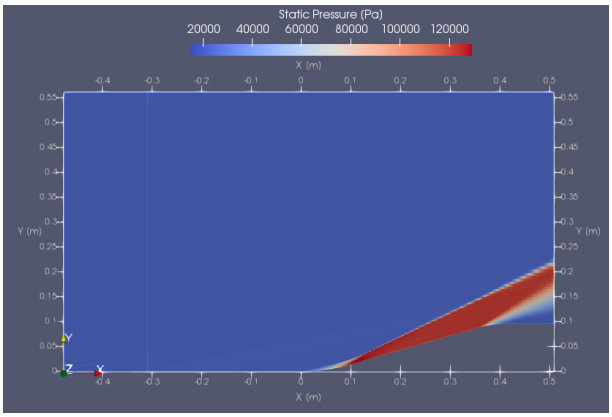


Figure 14: Wall static pressure.



(a) Mach number



(b) Static pressure

Figure 15: Iso-contours of Case 2 (hypersonic) and SST turbulence model.

project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

REFERENCES

[1] Guillermo Araya and Luciano Castillo. 2013. DNS of Turbulent Boundary Layers Subjected to Adverse Pressure Gradients, In Progress in Turbulence and Wind Energy IV, Martin Oberlack, Joachim Peinke, Alessandro Talamelli, Luciano Castillo, and Michael Hölling (Eds.). *Phys. Fluids* 141, 187–190. https://doi.org/10.1007/978-3-642-28968-2_39

[2] Guillermo Araya, Luciano Castillo, Charles Meneveau, and Kenneth Jansen. 2011. A dynamic multi-scale approach for turbulent inflow boundary conditions in spatially developing flows. *J. Fluid Mech.* 670 (March 2011), 581–605. https://doi.org/10.1017/S0022112010005616

[3] Peter Bradshaw. 1973. *Effects of streamline curvature on turbulent flow.* Technical Report AGARD-AG-169. Advisory Group for Aerospace Research and Development Paris (France).

[4] Tuncer Cebeci and Peter Bradshaw. 1977. *Momentum Transfer in Boundary Layers.* Hemisphere, Washington, DC.

[5] John F. Donovan, Eric F. Spina, and Alexander J. Smits. 1994. The structure of a supersonic turbulent boundary layer subjected to concave surface curvature. *Journal of Fluid Mechanics* 259 (January 1994), 1–24. https://doi.org/10.1017/S0022112094000017

[6] Walter G. Hoydysh and Victor Zakkay. 1969. An experimental investigation of hypersonic turbulent boundary layers in adverse pressure gradient. *AIAA Journal* 7, 1 (January 1969), 105–116. https://doi.org/10.2514/3.5042

[7] Mohan Jayaram, Margaret W. Taylor, and Alexander J. Smits. 1987. The response of a compressible turbulent boundary layer to short regions of concave surface curvature. *J. Fluid Mech.* 175 (February 1987), 343–362. https://doi.org/10.1017/S0022112087000429

[8] A. J. Laderman. 1980. Adverse Pressure Gradient Effects on Supersonic Boundary-Layer Turbulence. *AIAA Journal* 18, 10 (October 1980), 1186–1195. https://doi.org/10.2514/3.50870

[9] George H McLafferty and Robert E Barber. 1962. The effect of adverse pressure gradients on the characteristics of turbulent boundary layers in supersonic streams. *J. Aeronaut. Sci.* 29, 1 (January 1962), 1–10. https://doi.org/10.2514/8.9294

[10] Florian R. Menter. 2009. Review of the shear-stress transport turbulence model experience from an industrial perspective. *Int. J. of Computational Fluid Dynamics* 23, 4 (May 2009), 305–316. https://doi.org/10.1080/10618560902773387

[11] H. Reichardt. 1951. Complete representation of the turbulent velocity distribution in smooth pipes. *Z. Angew. Math. Mech.* 31, 7 (January 1951), 208–219. https://doi.org/10.1002/zamm.19510310704

[12] John D. Schmisseur. 2015. Hypersonics into the 21st century: A perspective on AFOSR-sponsored research in aerothermodynamics. *Progress in Aerospace Sciences* 72 (January 2015), 3 – 16. https://doi.org/10.1016/j.paerosci.2014.09.009

[13] Eric F. Spina, Alexander J. Smits, and Stephen K. Robinson. 1994. The Physics of Supersonic Turbulent Boundary Layers. *Annual Review of Fluid Mechanics* 26, 1 (January 1994), 287–319. https://doi.org/10.1146/annurev.fl.26.010194.001443

[14] Walter B. Sturek and James E. Danberg. 1972. Supersonic Turbulent Boundary Layer in Adverse Pressure Gradient. Part I: The Experiment. *AIAA Journal* 10, 4 (April 1972), 475–480. https://doi.org/10.2514/3.50122

[15] Walter B. Sturek and James E. Danberg. 1972. Supersonic Turbulent Boundary Layer in Adverse Pressure Gradient. Part II: Data Analysis. *AIAA Journal* 10, 5 (May 1972), 630–635. https://doi.org/10.2514/3.50167

[16] M. W. Taylor and A. J. Smits. 1984. The effect of a short region of concave curvature on a supersonic turbulent boundary layer. *22nd Aerospace Sciences Meeting* (January 1984). https://doi.org/10.2514/6.1984-169

[17] H. Thomann. 1968. Effect of streamwise wall curvature on heat transfer in a turbulent boundary layer. *J. Fluid Mech.* 33, 2 (August 1968), 283–292. https://doi.org/10.1017/S0022112068001308

[18] David C. Wilcox. 2006. *Turbulence Modeling for CFD* (3rd ed.). D C W Industries, La Canada, California.

Performance Evaluation of Monte Carlo Based Ray Tracer

Ayobami Ephraim Adewale
University of Tartu
Tartu, Estonia
adewale@ut.ee

ABSTRACT

The main objective of computer graphics is to effectively depict an image in a virtual scene in its realistic form within a reasonable amount of time. This paper discusses two different ray tracing techniques and the performance evaluation of the serial and parallel implementation of ray tracing, which in its serial form is known to be computational intensive and costly for previous computers. The parallel implementation was achieved using OpenMP with C++, and the maximum speedup was ten times that of the serial implementation. The experiment in this paper can be used to teach high-performance computing students the benefits of multi-threading in computationally intensive algorithms and the benefits of parallel programming.

KEYWORDS

ray tracing, Monte Carlo, Open Multi-Processing (OpenMP), high performance computing, algorithm, cluster, parallel programming

1 INTRODUCTION

Light tracing is an important aspect of computer graphics that has over the years been adopted to simulate the real life behavior of illuminance on an object, environment, or scene in different areas such as animations, games, and image rendering. The computation of this illumination is being done by computer programs which calculate illuminance on a particular scene relatively precisely [10]. Due to its importance, there have been different techniques adopted for this purpose, but the two most popular are rasterization and ray tracing [3].

In rasterization, an image in a vector format is taken and converted into a pixelated image known as a raster image for output on a video display or static environment. In ray tracing, an image is rendered by tracing the trajectories of light rays projected through pixels in a view plane [6]. The main differences between these two techniques is the way an image is rendered and the time it takes to render an image. Rasterization has been the most popular of the two, because it is faster and balances the performance needed with the ability to create acceptable images. Regardless of its advantage, the application of rasterization is limited in computer graphics when a photo-realistic image is needed. This is due to its poor handling of light reflection when rendering 2D and 3D images.

https://doi.org/10.22369/issn.2153-4136/12/1/4

Rendering an image with ray tracing is slow when compared to rasterization. However, it is able to render a more photo-realistic image, due to its good handling of shadows, reflections, and blurs. With the arrival of parallel programming and GPUs, significant performance gains have been achieved when rendering with the ray tracing technique. The ray tracing technique is divided into two types: traditional ray tracing and distributed ray tracing. Distributed ray tracing tries to extend the traditional ray tracing technique by sampling more rays than the number of pixels in the image. An example of traditional ray tracing is the Turner Whitted technique, while the Monte Carlo technique is an example of distributed ray tracing [11].

With recent advances in computing power, various researchers have picked up an interest in ray tracing. The main objective is to find different ways of reducing time to solution while retaining the property of creating realistic images. The use of parallel frameworks such as the Message Passing Interface (MPI) and Open Multi-Processing (OpenMP) have been proposed by researchers over the years. In this paper, a parallel implementation and a serial implementation of a ray tracing algorithm were studied based on rendering performance. The serial implementation was derived from a parallel implementation that was developed using C++ and made parallel using OpenMP [1]. The performance evaluation was carried out on two high performance computing (HPC) clusters provided by University of Tartu High Performance Computing Center [9].

The paper is organized as follows. Section 2 introduces the concept of ray tracing and OpenMP. Section 3 discusses different state-of-the-art methods. Section 4 discusses the parallel implementation with OpenMP in [1]. In Section 5, the result of the empirical analysis is presented and discussed. Finally, in Section 6, the conclusion and reflection are discussed.

2 CONCEPTS

2.1 Ray Tracing

Ray tracing takes an image from a 3D scene by tracing the trajectories of light rays through pixels in a view plane. Light tracing in ray tracing can be done in two ways: forward tracing and backward tracing. In forward tracing, rays are traced from eyes to the light source, while in backward tracing, rays are traced from the light source to the eyes. This method is compute intensive, because each ray emitted by the light source has to be traced to the eyes, even those that were not able to reach the eyes.

A typical ray tracing environment is made up of eyes or a camera, a scene, object(s), and a light source. Figure 1 describes a simple ray tracing scenario. A ray from the eyes is projected in a straight line for each pixel of the view plane into the scene. Ray intersection is checked to see if the ray intersects with any object in the scene.

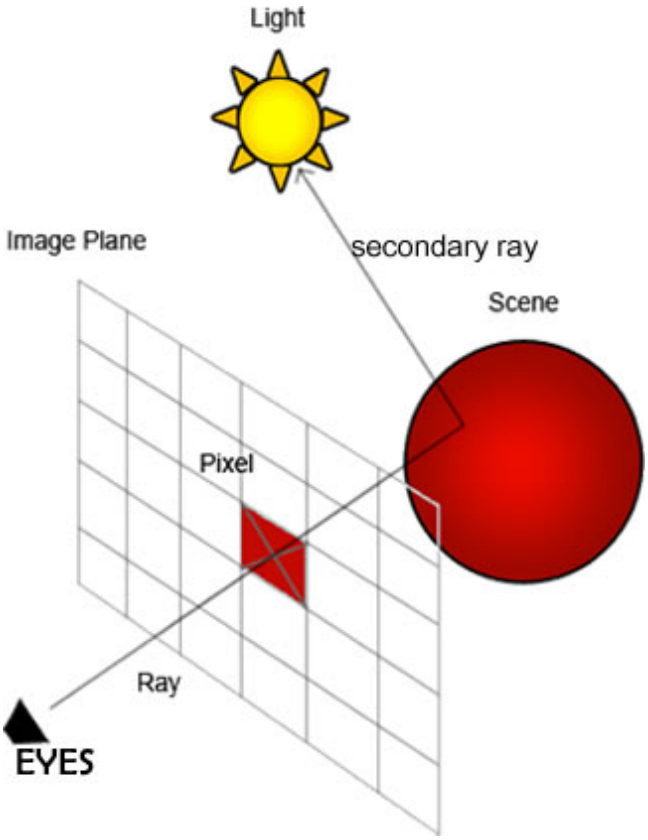


Figure 1: Ray Tracing Scenario.

This first ray is referred to as the primary ray, and its purpose is to discover the objects in the scene. There are three intersection possibilities for this ray, and Figure 2 depicts a simple scenario.

- No Intersection.
- One Intersection.
- Two Intersections.

The third case only happens if the object in the scene is opaque. If an intersection exists, a second ray referred to as the secondary ray is sent from the point of intersection to the light source. If this secondary ray that was emitted is blocked by an object in the scene, the color of the object is projected as a shadow, and if it successfully reaches the light source, that pixel is lit up. The lighting of the pixel is determined by this secondary ray. If there are two points of intersection, the distances of the two points to the eyes are calculated, and the closest point is selected. In ray tracing, the secondary ray is divided into three: shadow ray, reflection ray and refractive ray. These rays make it possible for the ray tracing technique to create a more realistic rendering.

Algorithm 1 is a simple ray tracer algorithm as described by Turner Whitted. In the algorithm, each ray cast through the pixel in the plane can be represented as a line that has an origin represented with O and a direction represented with l. At any time, a point on the line or ray can be represented with

$$Point = O + (l * d) \tag{1}$$

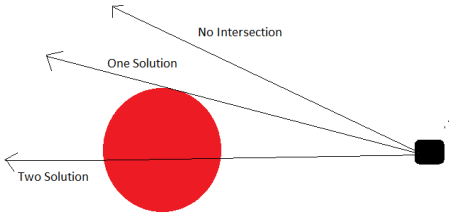


Figure 2: Ray Tracing Intersection Possibilities.

```
for each pixel in the viewing plane do
    for each object in the scene do
        if ray intersects an object in the scene then
            select min(d1,d2);
            recursively ray trace the reflection and refraction
              rays;
            calculate color;
        end
    end
end
```

Algorithm 1: Ray Tracing Algorithm

where d is the distance of the point from the ray origin. Having declared the origin and direction of the ray and also the location of the sphere, we proceed to know if a ray projected from the eye through a pixel in the view plane intersects with any sphere in our scene at any points. say, P_0, and P_1.

To do this, we need to solve for dc:

$$dc = C_0 \quad . \quad l; \quad where \quad C_0 = C - O. \tag{2}$$

dc is the distance of the projected ray from the center of the sphere to the ray origin, C is the center of the sphere, C_0 is the distance from the center of the sphere to the ray origin and . denotes a dot product. If dc is less than zero, then it can be assumed that there is no intersection, but if it is greater than zero, we proceed to the second step, which is to calculate the distance from the center of the sphere to the projected ray. This can be calculated using Pythagoras' Theorem, because we have now formed a right-angled triangle.

$$D = \sqrt{dc^2 - (C_0)^2} \tag{3}$$

If the value of D is greater than the radius of the sphere, then it means the ray does not intersect any point of the sphere, and we can move on to the next sphere. If the value of D is not greater than the radius, then we continue by looking for the points of intersection on the sphere. The point of intersection is calculated as:

$$P = O + (l * d_p) \tag{4}$$

where d_p is the distance from P to O, and it is calculated using:

$$d_p = dc - tc \tag{5}$$

where tc can be obtained from the second right-angled triangle in our sphere using Equation 6,

$$tc = \sqrt{r^2 - D^2} \tag{6}$$

where r is the radius of the sphere. If the ray intersects at two points, the distance of the second point represented as d_{p2} is calculated as $d_{p2} = dc + tc$.

Algorithm 2 describes the ray intersection algorithm.

FUNCTION: $boolean\ intersection(Ray * R,$
$Sphere * S, float * d_{p1}, float * d_{p2})$
$float\ C_0 = S \longrightarrow center - R \longrightarrow Ray\_Origin$
$floatdc = dot(C_0, R \longrightarrow direction)$
if $dc \leq 0.0$ **then**
  return $false$
else
  $float\ D = \sqrt{dc^2 - (C_0)^2}$
  if $D \geq S \longrightarrow radius$ **then**
    return $false$
  end if
  $float\ tc = \sqrt{(S \longrightarrow radius)^2 - D^2}$
  $d_p = dc - tc$
  $d_{p2} = dc + tc$
  return $true$;
end if

Algorithm 2: Ray Intersection Algorithm.

Merging Algorithms 1 and 2, we can derive the time complexity of a simple ray tracing algorithm as:

$$O(pnd) \tag{7}$$

where p is the number of pixels, which can be represented as:

$$pixels = width * height \tag{8}$$

and n is the number of objects in the scene. d is the time complexity of computing the intersection, which can be represented as 1. The time complexity can then be rewritten as:

$$O(whn) \tag{9}$$

This time complexity means that given a screen resolution of x pixels, a scene with a large number of objects will take more rendering time than a scene with fewer objects.

2.2 Distributed ray tracing

The major drawback of the traditional ray tracing method is that it causes aliasing. When an intersection is not found, the background color is returned, and this means that for every point, a color is always returned. This behaviour can always lead to rendering of unintended patterns. The solution to this is to introduce more rays into the scene and also more randomness.

Distributed ray tracing extends the traditional ray tracing method by introducing the concept of sampling to remove the aliasing effects that exist in traditional ray tracing. By removing the aliasing effects, a photo-realistic image with better shadows, reflections, and refraction is rendered. The single ray in traditional ray tracing is replaced with a distribution of rays, and an average of a random sampling of the rays is taken to reduce aliasing effects. These random samples can be generated using Monte Carlo (MC) or Quasi-Monte Carlo (QMC) algorithms. The rendering time of a distributed ray tracing then becomes a function of the number of random rays sampled. This means that the rendering time of a distributed ray tracing application is always more than that of a traditional ray tracing technique.

In traditional ray tracing, tracing of rays is usually terminated after reaching a diffuse surface, but in the distributed technique, after a ray hits a diffuse object, child rays are generated randomly according to the bi-directional reflection and refraction distribution function of the diffuse surface [5].

A simple MC-based distributed ray tracing algorithm is presented in Algorithm 3. The algorithm shows how random rays are distributed in a single pixel, and for each ray a computation is done. The MC algorithm is then used for sampling, and the actual color of the pixel is computed based on the sampled rays.

for *each pixel in the viewing plane* **do**
  for *each ray in random rays* **do**
    for *each object in the scene* **do**
      if *ray intersects an object in the scene* **then**
        select min(d1,d2);
        recursively ray trace the reflection and refraction rays;
        return calculated color;
      end
      if *no intersection* **then**
        return background color;
      end
    end
  end
  random sample rays with montecarlo;
  calculate color average;
end

Algorithm 3: Distributed Ray Tracing Algorithm.

The time complexity of the algorithm is given as:

$$O(whrn) \tag{10}$$

where r is the number of distributed rays projected to each pixel. This time complexity means that a distributed ray tracing algorithm using the same configurations as a traditional ray tracing algorithm will surely take more rendering time.

2.3 OpenMP

Open Multi-Processing (OpenMP) has often been referred to as the de-facto standard for writing parallel programs with shared memory architectures, and these parallel programs can simply be achieved by adding compiler pragmas to the serial equivalent [7]. It allows for shared memory parallel programming in languages like C, C++, and Fortran. With OpenMP being a shared memory architecture, all threads spawned out have access to the same main memory and the same data. OpenMP is often used when there is a need to facilitate the execution of legacy code on a multi-core processor in order to utilize all its cores [7].

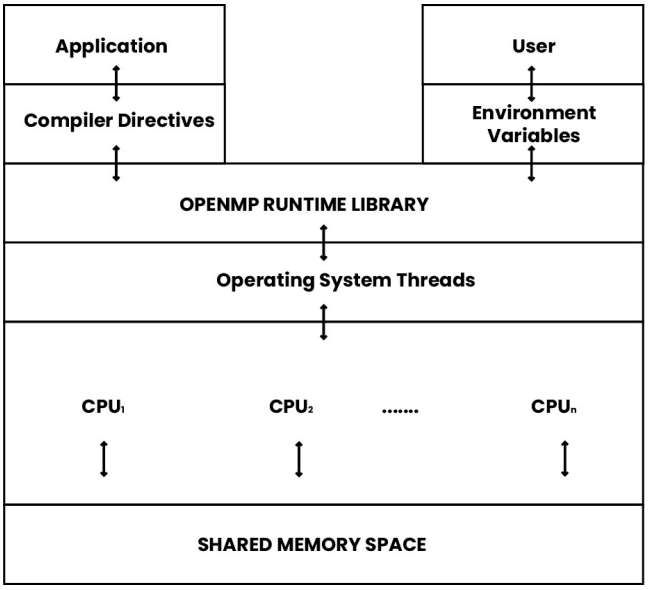


Figure 3: OpenMP Architecture.

In Figure 3 an architecture of OpenMP is presented. It can be seen that all the operating system threads have access to the same shared memory space, and the threads are being managed by the OpenMP run-time library. The parallelism in the run-time library is then specified using compiler directives, which are available in the application code.

In OpenMP, threads run in parallel, execute the same code, and share the same memory space. Each thread spawned out has a unique identifier that can be obtained using *omp_get_thread_num*().

Table 1 gives a list of some useful OpenMP directives and an explanation of what they do. The directives are always defined at the beginning of a blocked region. The *#pragma omp parallel* marks the entry point of a parallel region, and without that, threads cannot be spawned out.

A simple OpenMP algorithm that prints to console *hello world* is presented in Algorithm 4. The algorithm starts by specifying the parallel region of the code with the *#pragma omp parallel* directive, and the default number of threads is used. The blocked region is executed by all the threads in parallel. Each thread gets its identifier and prints *hello world* plus the identifier of the thread.

In summary, there are five major elements of parallel programming with OpenMP:

- Create threads with shared memory.
- Loop parallelism.
- Nested parallelism.
- Dynamic task scheduling.
- Thread synchronization.

3 RELATED WORK

Different state of the art methods have been developed to guarantee faster ray tracing rendering in computer graphics. Some of the most popular methods have explored the application of parallel programming techniques and the use of fast computation hardware like GPUs. The use of the Message Processing Interface (MPI) was explored in [2]. MPI is a message passing library which allows all available processors to communicate while executing the same program. All processors execute the same code, but in a separate isolated memory spaces, and communication between each processor is handled with an API. In the research, a near linear relationship between the number of processors used and the ray tracing rendering rate was reported. It was also reported that the efficiency rate achieved after adapting the serial ray tracing code to use MPI was over 98%. Unlike in [2], the focus of this paper is in the use of a shared memory device and not a distributed memory device which is used by MPI. Also, unlike in OpenMP, the time it took to communicate between processors used in MPI was also factored in when calculating the ray tracing rendering time.

Also in [6], an empirical study on the use of both OpenMP and MPI was explored for creating a parallel ray tracing. In the report, a pixel-wise load balancing scheme was introduced to allow load

Table 1: OpenMP Directives.

Directive	Function
#pragma omp parallel	Specifies the parallel region of a code.
#pragma omp for	Defines the start of a loop parallelism.
#pragma omp for simd	Defines the start of a loop parallelism that uses SIMD instructions.
#pragma omp single	Specifies a region that should be executed by a single thread.
#pragma omp sections	A non-iterative shared parallel section.
#pragma omp master	Specifies a region that should be executed only by the main thread.
#pragma omp ordered	Specifies a region that must be executed in order.
export KMP_AFFINITY=value	Used to define thread affinity types and is specific to Intel compilers. *value* can be *verbose*, *scatter*, or *compact*.
omp_get_num_threads	Used to get the count of all currently running threads.
omp_get_thread_num	Used to get the identifier of a currently running thread.
omp_set_num_threads	Used to set the number of threads that should be used for executing parallel regions.

```
#pragma omp parallel;
if main then
    int id = omp_get_thread_num();
    print("hello (%d)", id);
    print("world (%d) ", id);
end
```

Algorithm 4: Simple OpenMP Hello world.

distribution for some specific scenes. The report proved that both OpenMP and MPI are capable of achieving near optimal efficiency when used in ray tracing. In the research, a near linear speedup was also reported for both methods. Also, compiler performance between an Intel compiler and a GNU C++ compiler were studied, and the research reported that the Intel compiler outperformed the GNU C++ compiler. Unlike in [6], in our paper, the effect of different thread affinity types and different distributed systems was explored.

The use of GPU clusters for ray tracing rendering was explored by *Chen et al* [4]. In the research, the researchers were able to use GPU clusters to improve the rendering performance of a real-time ray tracing. The frame per second (FPS) achieved by the GPU cluster was compared to a single node GPU and a CPU. The research showed that the GPU cluster achieved more FPS than a single node GPU, and the single node GPU achieved more FPS than the CPU. A massively parallel ray tracing algorithm using a GPU was also developed by *Qin et al* [8]. The research showed that the GPU significantly reduced the rendering time of the ray tracing algorithm.

4 METHODS

Distributed ray tracing is computationally intensive, because intensive ray-geometry intersection computation must be done for rays projected into the scene through each pixel of the view plane. Since each ray is not dependent on the other, this also makes ray tracing embarrassingly parallel. To show that a distributed ray tracing is embarrassingly parallel, a parallel implementation in [1] was used as a case study. The scene consists of a spherical light source, a glass, a mirror, and one Cornell box, which was made of 6 spheres.

A loop-level parallelism was introduced using OpenMP to achieve parallelism in [1]. The loop-level parallelism made it possible to divide the intensive ray-geometry intersection among all participating threads. Each thread created by the process is allocated a task, they execute their own part of the code, and return with the result. Since OpenMP is a shared memory framework, the scene to be rendered was placed in shared memory in the form of a data structure, and this eliminated the possible overhead that could be introduced during data communication between threads, as all threads could access the same scene data.

In Algorithm 5, it can be seen that two levels of loop-level parallelism were implemented: one for each pixel and another for each ray distributed into the pixel. This was done because each pixel is independent of the others, and the same can be said for each of the distributed rays. Based on the algorithm, an empirical analysis is then carried out on how compiler types and numbers of threads affect rendering time. The focus here is measuring scability using speedup. Speedup was calculated using the formula:

$$Speedup = T_s/T_p \tag{11}$$

where T_s is the rendering time in serial and T_p is the rendering time in parallel for different thread numbers.

Also, the effect of using different thread affinity was explored in this paper. With thread affinity, we are able to control OpenMP thread placement, and this allows us to study the effect on the parallel ray tracing algorithm. In this paper, two types of thread affinity were explored, namely Scatter and Compact. When the Compact thread affinity type is specified, all the spawned out threads are placed close to each other. With Scatter, the threads are distributed as evenly as possible, and this eventually reduces the cache and memory bandwidth contention between threads.

```
#pragma omp parallel;
#pragma omp for;
for each pixel in the viewing plane do
    #pragma omp for;
    omp_set_num_threads(thread_num);
    for each ray in random rays do
        for each object in the scene do
            if ray intersects an object in the scene then
                select min(d1,d2);
                recursively ray trace the reflection and
                 refraction rays;
                return calculated color;
            end
            if no intersection then
                return background color;
            end
        end
    end
    random sample rays with montecarlo;
    calculate color average;
end
```

Algorithm 5: Parallel Ray Tracing with OpenMP.

5 RESULTS AND DISCUSSION

The code takes a single parameter as an input, which is the number of samples per pixel (spp), and for the performance evaluation of the distributed ray tracing, 25,000 samples per pixel (pixel) were used. The experiment was done on HPC clusters provided by University of Tartu [9], and the configuration is presented in Table 2.

Table 2: HPC Configuration.

Rocket cluster	Vedur cluster
20 cores	32 cores
2x Intel Xeon	2x AMD Opteron
64GB RAM	150GB RAM
1TB hard disk drive	500GB hard disk drive
4x QDR Infiniband	4x QDR Infiniband

The serial implementation of the parallel code in [1] was compiled using two different compilers: g++ compiler and Intel C++ (ICC) compiler. Figure 4 shows the rendered scene when the serial implementation was executed. The time taken to render the scene was measured, and the result is presented in Table 3.

Figure 5 is the scene rendered when the parallel implementation in [1] was executed, and it can be seen that the quality of the scene was preserved.

The rendering time of the parallel implementation was measured for both the Intel and g++ compilers for 2^n threads, where n is 1, 2, 3 or 4. Due to the number of cores available on Rocket Cluster, the

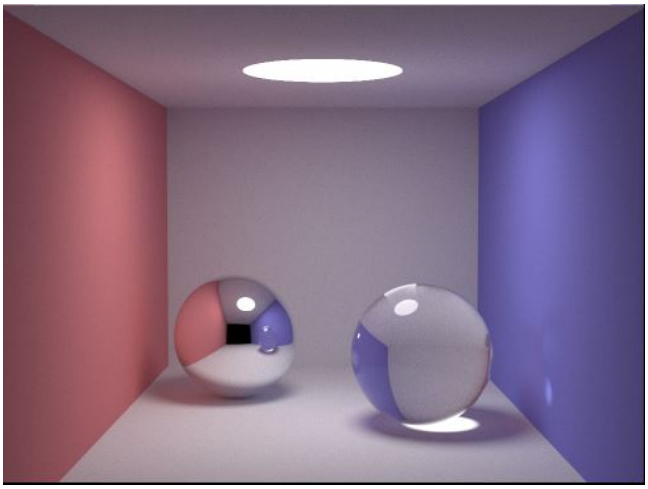


Figure 4: Serial ray tracing image with Cornell box, one light source, and two spheres, making use of 25,000 samples per pixel.

Table 3: Serial Ray Rendering.

Compiler	Rendering time(minutes)
Intel ICC compiler	120
g++ compiler	182

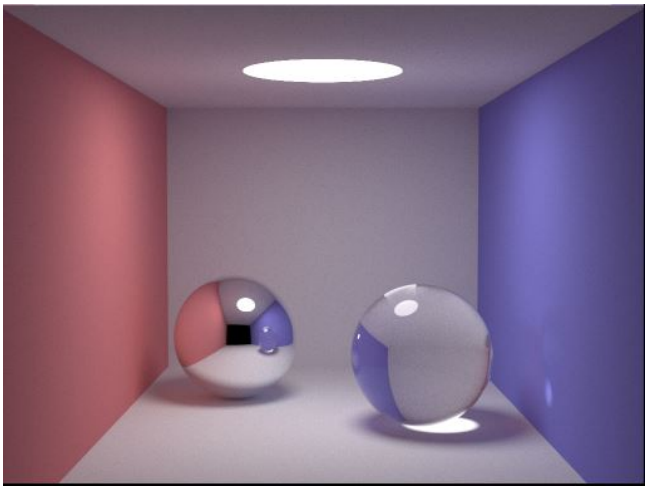


Figure 5: 20 threads on an Intel Xeon based cluster (Rocket cluster).

last thread number used during performance testing was 20 threads. Since the main aim of parallelism is to speed up the performance while maintaining the quality of the image, the quality of the image was studied throughout the test, and it was observed that the quality was preserved for each thread range.

Table 4 shows the rendering time in seconds and the speedup after optimizing using OpenMP for different thread numbers while executing with the Intel compiler. In Table 4, it can be seen that a linear speedup was achieved in the parallel implementation, and this is because the ray tracing algorithm is an embarrassingly parallel algorithm.

Table 4: Intel Xeon based cluster with Intel compiler

Threads	Rendering Time	Speedup
2	60 minutes	2.0
4	33 minutes	4.0
8	17 minutes	7.0
16	8 minutes	14
20	7 minutes	18

Table 5: Intel Xeon based cluster with g++ compiler

Threads	Rendering Time	Speedup
2	86 minutes	2.0
4	46 minutes	4.0
8	24 minutes	8.0
16	12 minutes	15
20	10 minutes 20 seconds	14

Table 5 shows the rendering time and speedup with the g++ compiler in the Rocket cluster. It is evident that the Intel compiler rendered the image faster than the g++ compiler; however, they both achieved a linear speedup for up to 16 threads.

The impact of using different thread affinities was also measured, and this result can be seen in Figure 6, which shows that the two thread affinities only increased the efficiency of the parallel execution of the code. We can also see that the speedup on 20 threads when using the compact thread affinity reduced compared to the scatter thread affinity.

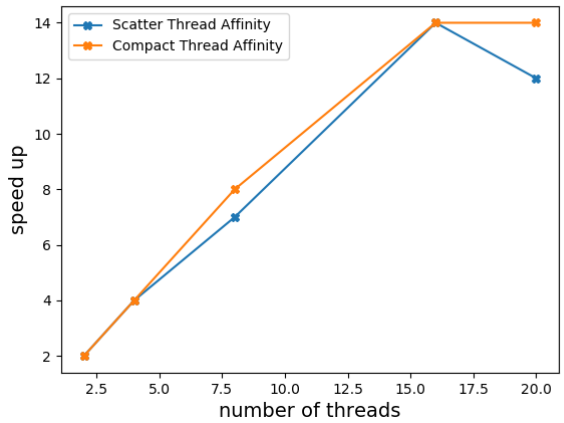


Figure 6: Speedup for Thread Affinity (Compact and Scatter)

After computing the performance of the parallel code on the Intel Xeon cluster, attempts were made to see the performance of the parallel code on an AMD based cluster, which is also one of the University of Tartu clusters. This was done because the AMD based cluster is not limited to 20 cores on a single node like the Intel Xeon cluster, but rather 32 cores. The rendering time of the parallel implementation was measured on only the Intel compiler, and the serial code rendering time was 252 minutes (4 hours 21 minutes).

Table 6: AMD based cluster with Intel compiler.

Threads	Rendering Time	Speedup
2	126 minutes	2.0
4	62 minutes	4.0
8	31 minutes	8.0
16	16 minutes	16
20	10 minutes	25

Comparing the AMD based cluster performance to that of the Intel Xeon based cluster, it can be observed that there is a difference in the rendering time, and this is due to the difference in hardware architecture of the two clusters. It is evident that linear speedup was also achieved up to 20 threads, and 100% efficiency was achieved up to 16 threads when running the parallel code on the AMD cluster. Again, the quality of image rendered was the same across the thread range.

In Figure 7 and Figure 8, the graphical representations for rendering time and speedup of all three cases are presented. It can be seen in Figure 7 that the rendering time decreases as the number of threads increases. However, as the number of threads approaches 20, the difference in rendering time starts to decrease for all cases. In Figure 8, the graphical representation of the speedup is presented. It can be seen that the speedup for each number of threads is specific to each case. However, the speedup achieved is the same when the number of threads spawned is less than five.

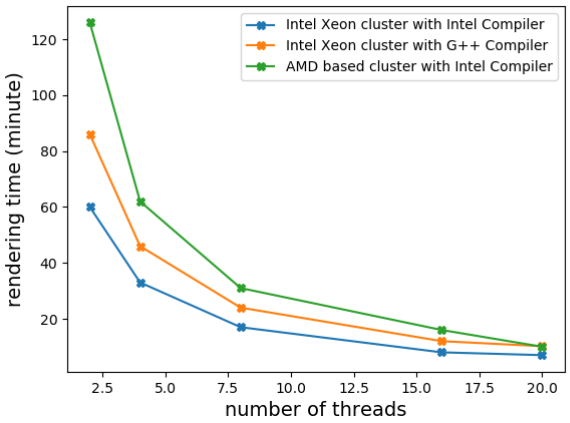


Figure 7: Render Time.

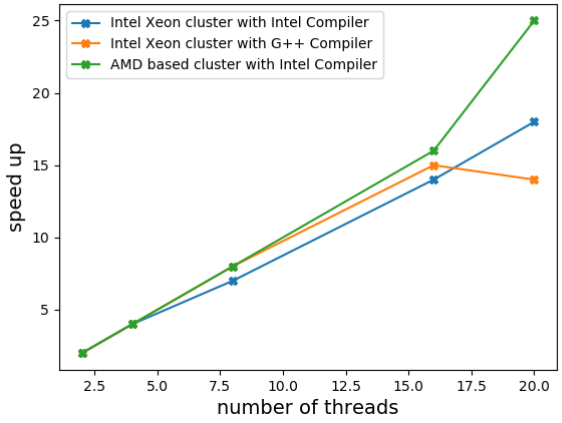


Figure 8: Speedup.

6 CONCLUSION AND REFLECTION

6.1 Conclusion

In this paper, a serial Monte Carlo based distributed ray tracing technique derived from a parallel implementation in [1] was compared to its parallel implementation, and the performance speedup on two compilers was measured while changing the number of threads. The test was carried out on University of Tartu's HPC cluster, and the performance test showed that a linear speedup was achieved, and while using thread affinities of type compact and scattered, an 100% efficiency was achieved for different thread numbers.

A very interesting future work will be to compare a serial real time distributed ray tracing implementation with a parallel implementation of both OpenMP and MPI. Since MPI is a distributed memory framework, overhead due to data communication between processors can be anticipated in MPI, and this might make it inefficient for pluralizing scenes with large data sets, such as data sets used in real time ray tracing.

6.2 Reflection

The experiment carried out in this paper was done while taking the parallel computing course at University of Tartu, and it was my first attempt at working in a high performance computing environment. Prior to this, I had no knowledge of parallel programming nor parallel computing. The greatest challenge for me while working on this project was understanding different frameworks for writing parallel programming codes like OpenMP and MPI. At the end, this experiment introduced me to parallel computing, different techniques for writing parallel programs, and methods for calculating parallel efficiency. In addition, it enriched my experience in the management and allocation of resources in leading-edge computing infrastructure through writing Slurm scripts.

Overall, working on this project was a unique experience and opportunity for me in the field of HPC. The experience gained was also useful when I was writing machine learning codes for my master's thesis.

ACKNOWLEDGMENTS

Huge thanks go to University of Tartu for making their HPC clusters available for this experiment. Also, big thanks go to Dr. Benson Muite for his support and guidance throughout the course of the experiment.

REFERENCES

[1] Kevin Beason. 2013. Monte Carlo Global illumination Code in C++. Retrieved April 4, 2020 from http://www.kevinbeason.com/smallpt/

[2] Charles B. Cameron. 2008. Parallel Ray Tracing Using the Message Passing Interface. *IEEE Transactions on Instrumentation and Measurement* 57, 2 (2008), 228–234.

[3] Chun-Fa Chang, Kuan-Wei Chen, and Chin-Chien Chuang. 2015. Performance comparison of rasterization-based graphics pipeline and ray tracing on GPU shaders. In *2015 IEEE International Conference on Digital Signal Processing (DSP)*. 120–123. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7251842

[4] M. Chen and H. Huang. 2018. A Real-time Parallel Ray-tracing Method Based On GPU Cluster. In *2018 IEEE International Conference of Safety Produce Informatization (IICSPI)*. 327–330.

[5] Balázs Csébfalvi. 1997. A Review of Monte Carlo Ray Tracing Methods. Retrieved April 4, 2020 from http://www.cescg.org/CESCG97/csebfalvi/index.html

[6] Sadraddin A. Kadir and Tazrian Khan. 2008. *Parallel Ray Tracing using MPI and OpenMP*. Technical Report. Stockholm, Sweden.

[7] Manji Mathews and Jisha P. Abraham. 2016. Automatic Code Parallelization with OpenMP task constructs. In *2016 International Conference on Information Science (ICIS)*. 233–238.

[8] Y. Qin, J. Lin, and X. Huang. 2015. Massively parallel ray tracing algorithm using GPU. In *2015 Science and Information Conference (SAI)*. 699–703.

[9] University of Tartu. 2014. HPC cluster resources. Retrieved May 29, 2020 from https://hpc.ut.ee/en/resources/

[10] Jan Škoda and Martin Motyčka. 2018. Lighting Design Using Ray Tracing. In *2018 VII. Lighting Conference of the Visegrad Countries (Lumen V4)*. IEEE, 1–5. https://ieeexplore.ieee.org/document/8521111

[11] T. Whitted. 2020. Origins of Global Illumination. *IEEE Computer Graphics and Applications* 40, 1 (2020), 20–27.

Training Neural Networks to Accurately Determine Energies of Structures Outside of the Training Set Using Agglomerative Clustering

Carlos A. Barragan
Department of Chemistry and Biochemistry
California State University, Fullerton
Fullerton, CA
carlosb@csu.fullerton.edu

Michael N. Groves
Department of Chemistry and Biochemistry
California State University, Fullerton
Fullerton, CA
mgroves@fullerton.edu

ABSTRACT
Machine learning has accounted for solving a cascade of data in an efficient and timely manner including as an alternative molecular calculator to replace more expensive *ab initio* techniques. Neural networks (NN) are the most predictive for new cases that are similar to examples in their training sets; however, it is sometimes necessary for the NN to accurately evaluate structures not in its training set. In this project, we quantify how clustering a training set into groups with similar geometric motifs can be used to train a NN so that it can accurately determine the energies of structures not in the training set. This was accomplished by generating over 800 C_8H_7N structures, relaxing them using DFTB+, and grouping them using agglomerative clustering. Some of these groups were assigned to the training group and used to train a NN using the pre-existing Atomistic Machine-learning Package (AMP) [10]. The remaining groups were evaluated using the trained NN and compared to the DFTB+ energy. These two energies were plotted and fitted to a straight line where higher R^2 values correspond to the NN more accurately predicting the energies of structures not in its training set. This process was repeated systematically with a different number of nodes and hidden layers. It was found that for limited NN architectures, the NN did a poor job predicting structures outside of its training set. This was improved by adding hidden layers and nodes as well as increasing the size of the training set.

Categories and Subject Descriptors
Computing methodologies - *Machine learning, Machine learning approaches, Neural networks*

General Terms
Algorithms, Measurement, Reliability

Keywords
Atomistic Machine-learning Package, neural network, genetic algorithm, agglomerative hierarchical clustering, Density Functional Tight Binding.

DOI: https://doi.org/10.22369/issn.2153-4136/12/1/5

1. INTRODUCTION
Machine learning programs are becoming increasingly popular and are a form of widely-accepted method for calculating properties. Such techniques are readily available in any field to help solve problems that would be otherwise difficult to solve or envision. An example of their capabilities is when a research group known as Laser Interferometer Gravitational-wave Observatory (LIGO) witnessed the phenomenon of gravitational waves in outer space. They imposed a technique known as Deep Learning which can learn from immense raw data using artificial neurons or neural networks [5]. Machine learning techniques can work to closely resemble atomistic calculators.

The common approach that we observe when preparing data to train and test a neural network (NN) — such as images or atomic descriptions — is to randomly assign data to the train and test sets. This strategy is appropriate when it is not expected that the NN will need to make predictions on test candidates that are very different from what it was trained on. One example of the random strategy is when researchers prepared tens of thousands of randomly plausible molecules to understand the relationship between light-harvesting systems and excitation energy transfer times such as those found in the pigments of plants [6]. The excitation transfer time refers to how pigments can transfer energy over long distances in the presence of a light-harvesting system, such as light from the sun, to produce energy. In this simulation, machine learning techniques were used to reduce computational cost and to discover which chromophoric molecule (or excitation system) had the most efficient transfer time. Another example of this strategy is using machine learning to discover drug designs in the field of medical science [14]. However, even in the light-harvesting system and excitation study, an improved method is preferred, because the random method is not evenly-sampled and could have redundant information.

We discuss the widely used random approach to introduce an alternative: clustering the data and training the NN with some of the clusters and testing the NN with the remaining clusters. In this case, clustering the molecules organizes them into groups of similar motifs [17]. By training the NN with a group of clusters and then testing it on another group of clusters, the ability for the NN to predict structures outside of its training set can be quantified. This paper will show that when the clustering method is implemented, the predictive ability of the trained NN will dramatically decrease relative to the random approach, but as the NN architecture grows, the NN becomes better at accurately predicting the potential energy of the trial system. This demonstrates that the clustering method can help to define how robust a trained NN is to predict properties of structures not in its training set.

In this work, we present that NN architectures and the training set size are both vital components when applying a NN to structures that are unique from the training set. By clustering the training set, the robustness of the NN can be quantified. When compared to randomly selecting the training and testing groups, it is clear how clustering can help guide designing the structure of a NN so that it can be used for a wider array of applications. In the Methods section, we present how NN architectures and the training sets were assembled using clusters. NNs are trained systematically with three, four, and five hidden layers, each with 10–35 nodes. In the Results section, we present plots of R^2 values from the linear fit between the relaxed DFTB+ energies and the NN calculated energies. Finally, we remark on how clustering training data can help to ensure NNs are robust and guide the architecture of an effective NN.

2. Literature Review

Atomistic calculators have become a beneficial tool for calculating properties at the atomic scale. Calculations start with descriptions of the physical interactions of the atoms, which then yields information about their collective behavior [16]. The Density Functional based Tight Binding (DFTB+) is an example of such atomistic calculator. Through the DFTB+ package, we collected potential energy values of readily-formed molecules. Atomistic calculators have helped perform calculations when studying computational techniques, including the genetic algorithm (GA).

A GA is a computational heuristic that can be used to solve for the global atomic minimum of chemical structures. The GA is efficient at searching through many different molecular motifs in the potential energy surface by utilizing the principles of natural selection. In nature, a species must increase its fitness if it is going to learn to adapt to its environment [18]. The GA is designed to work in a similar manner, because it will iteratively improve upon the immediate population [8]. This is because in a real-world situation the GA does not know the answer. Neither the configuration space nor the global minimum are known prior to the search [8]. The evaluation step — the aspect of the GA where the lowest conformation energy is being searched — will be the most time-consuming for the optimization of offspring structures. Typically, structures are relaxed using an expensive *ab initio* method.

The GA begins with two parent molecules in a starting population. The cut-and-splice operator cuts the parent molecules in two, resulting in four fragments. Typically, a fragment from each is selected at random to be spliced together to form a new child molecule [1]. The cut-and-splice apparatus is illustrated on the upper right-hand corner of Figure 1, where the box-like figures are meant to represent molecular sites. Two parent molecules are shown with the same number of boxes to represent their similar chemical stoichiometry. Upon performing the cut, the parent molecules are now color-coded to signify their fragments (parent molecule 1 is shown in blue and green while parent molecule 2 is shown in red and yellow). Finally, two fragments are collected from each parent molecule and spliced together to form a new structure. This new structure is the child structure (shown in blue and yellow) and is one plausible combination of bringing in genetic information from both parent molecules. Given that the program will read a diverse molecular population, the outcome of the cut-and-splice operation will be different combinations of diverse offspring structures.

The next step of the GA heuristic is the evaluation process. The energy of the child structure is evaluated, and if the structure is fit enough to be in the population, it is added. The search for the lowest conformation energy then continues with a new parent pair. The structure with the local minimum is in the bottom-center of Figure 1. In the iterative process of the GA, potential energies of new offspring structures are constantly being compared to the energy of the population [9]. Molecular configurations that are better in terms of minimum potential value replace configurations with a higher potential energy. If the energy of the offspring structure is lower than the energy of the population, then it is added to the population. Conversely, if the potential energy of the child structure is higher than the energy of the population, then that offspring molecule is deleted. The program thus continues the evolutionary-driven perspective of constantly searching for the lowest conformation energy while eliminating unfit offspring structures with higher energies. Given enough generations, the population will eventually get trapped in either a local minimum or find the global minimum.

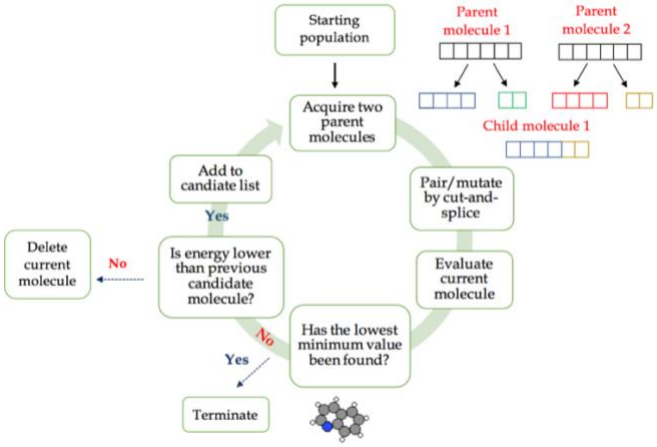


Figure 1. A flowchart outlining the steps of the GA from start to end. The program begins with a starting population, which proceeds in a loop where two molecules are selected and a new candidate structure is created using a cut-and-splice operation. The candidate is then evaluated and potentially replaces the least favorable member of the population. The GA terminates upon locating the lowest conformation structure.

The GA can be integrated with a NN to improve its ability to search the configuration space for the global minimum [12]. Machine learning techniques have revolutionized data analysis in most things we use today, such as travel booking, navigation, media recommendation, image recognition, and competitive board games. These systems are driven by the computational power, significant amount of data, and training ability of the neural network [2]. The neural network is comprised of hidden layers and nodes. Figure 2 depicts a NN containing an input layer, two hidden layers with four nodes each, and an output layer. Each node passes information forward in the form of entities known as weights and constant biases which are calculated via an activation function [10]. The process of the NN works like the human brain. The NN relies on the structural connection of nodes, where information can be obtained and sent between hidden layers, like how the brain relies on the connection of neurons [11].

Figure 2 showcases a fast feed forward NN. The connection of the eight nodes and two hidden layers is represented by the gray-colored arrows. The input layer passes a weight value representing connection strength to the first, second, third, and fourth nodes of the first hidden layer. If we focus on the first node of the first hidden layer, we can witness how that information is then passed on to the

first, second, third, and fourth node of the second hidden layer. This process repeats in the second node of the first hidden layer to all nodes of the second hidden layer, and so forth, until all nodes are inter-connected. The NN can only advance in one direction (from left to right) and terminates with an output value with no loops or turns [15].

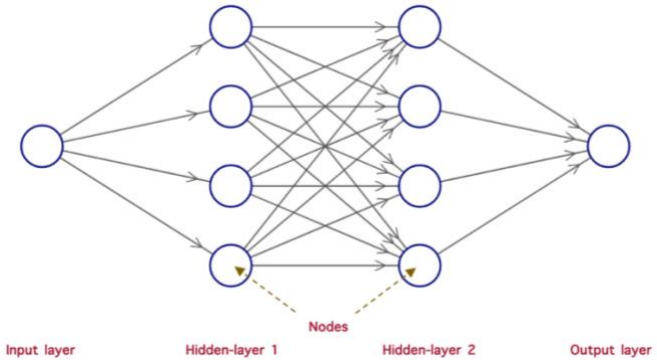


Figure 2. A visualization of a neural network containing an input layer, two hidden layers with four nodes each, and an output layer.

Numerous cluster programs exist, but the clustering method utilized for this project is known as agglomerative hierarchical clustering (AHC). Clusters created by AHC have been used efficiently on molecular populations to improve GA efficiency [9]. In general, molecules of the same chemical stoichiometry exist in different chemical configurations, such as lines, rings, or a combination of both. The AHC recognizes this diversity of structures and attempts to group them according to their close molecular similarity. This is achieved through the similarity threshold, which is the cutoff that groups structures together. What makes the AHC unique from other clustering programs is its bottom-up approach, where each data point starts in its own cluster and then proceeds by successfully merging similar classes of clusters together, which forms a hierarchy [7].

Figure 3 showcases this process through the visualization of a simple dendrogram. At first, each structure lives in its own cluster. When the AHC recognizes similar structures, it groups those two clusters (which were separated before) to form one cluster. At the end, the AHC has created three clusters (presented in blue, red, and green). Although seven molecules are shown in Figure 3, the AHC program can read in a variety of readily-formed chemical structures to form more sophisticated groups.

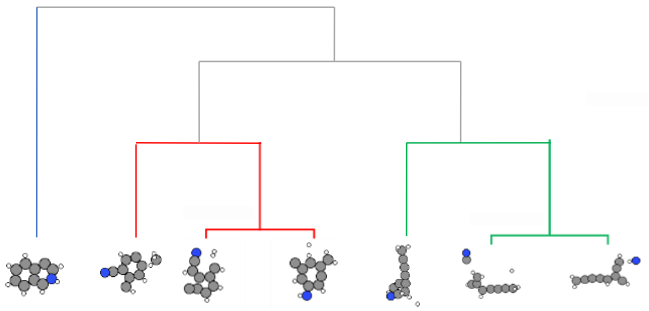


Figure 3. A segment of a dendrogram showing seven C_9H_7N molecules being grouped into three clusters (blue, red and green) based on a similarity index.

3. METHODS

To test how well the NN performs, we compared its values to those determined from DFTB+. DFTB+ combines the accuracy given from the DFT method with the efficiency of the Tight-Binding (TB) method [3]. This atomistic calculator was chosen because it is a fast, empirical method that allows us to perform many simulations with an appropriate level of accuracy. All structures are composed of C_9H_7N, whose global minimum is quinoline. This molecule is chosen because it is complex enough to have many local minima, but not so many that it becomes difficult to easily categorize them manually.

The NN was implemented using the Atomistic Machine-learning Package (AMP), an open-source, Python-based (accelerated by Fortran) code that was built to interface seamlessly with the Atomic Simulation Environment (ASE) [13]. ASE is an open-source, Python-based common front end that is capable of supporting many molecular calculators. AMP is capable of using several descriptors and activation functions for its NN; however, for this project, the default Gaussian descriptor and hyperbolic tangent activation function were used.

The comparison scheme comparing energies calculated by NN to those calculated by DFTB+ is illustrated in Figure 4. C_9H_7N molecules were evaluated with both NN and DFTB+ calculators. These energies were compared to determine how well a NN accurately calculated its energy. Once the testing set was fully evaluated, the NN and DFTB+ energies were plotted and fit to a straight line. The R^2 value from this fit was used to quantify how close the NN can represent the DFTB+ calculated value. The R^2 value determined from a data set where the train and test sets were identical was 0.9999, indicating a nearly perfect match. The lower the R^2 value, the less predictive the NN is on average at determining the energy calculated by DFTB+.

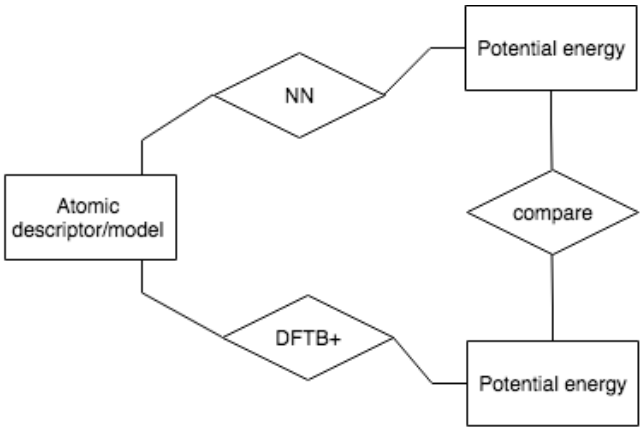


Figure 4. An overview of the comparison scheme in which NN and DFTB+ are used to evaluate the predictive nature of the NN. Both the NN and DFTB+ are used to evaluate the potential energy of a candidate, and the two values are compared.

A starting population consisted of 813 molecules generated from an evolutionary algorithm using the C_9H_7N stoichiometry. These were clustered into 21 clusters using AHC. The similarity index was arbitrarily set to generate 21 clusters, so that each cluster was sufficiently small, to have some flexibility in grouping them into five equally-sized groups with roughly 20% of the total structures in each group. Regardless of this effort, group 1 was composed of one very large cluster (288 molecules), while the remaining groups

were composed of roughly equal numbers of structures (133, 129, 125, and 138 molecules). Figure 5 illustrates a schematic showing group formation separated by arrows from left to right. The first step shows a sample of the population set. The second step shows that all similar molecules were clustered using the AHC method. The third and final step presents Groups 1–5 (represented by circles) which contain approximately 16% of the total population from the 813 molecules (Group 1 has 35% of the structures).

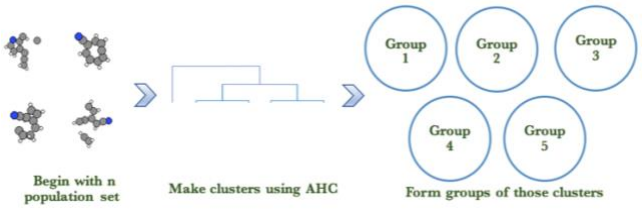


Figure 5. A schematic displaying how groups were formed from the results of the agglomerative clustering of initially generated structures. Each group consists of approximately 20% of the atomic molecules from the population.

The train/test cycle presented in Figure 6 outlines how the five train/test groups were created. When training a NN, a large training set is ideal. Sometimes this is not possible, so we tested the effect of 20% and 80% training set sizes to quantify the effect of a limited training set on a NN versus having a much larger training set. Figure 6 outlines the case of a 20% training set size. To start, one of the five groups was designated the training group, while the other four groups were combined to make the test set. For example, Test 1 included 525 molecules (the combined total of Groups 2–5) while Train 1 included 288 molecules. A NN was trained using the training set, and then all the DFTB+ energies of the structures in the test set were compared to those calculated using the newly-trained NN (as in Figure 4). This process was repeated four more times, where each of the five groups had a turn being the training set. The purpose of the train/test cycle was to average out any structure-related issues in any of the training sets. For each of the five sets, a scatter plot of the NN energy versus the DFTB+ energy was fit to a straight line, and the five R^2 values were averaged. To calculate the results from 80% of the structures being in the training, the process in Figure 6 was repeated with the training and testing groups reversed.

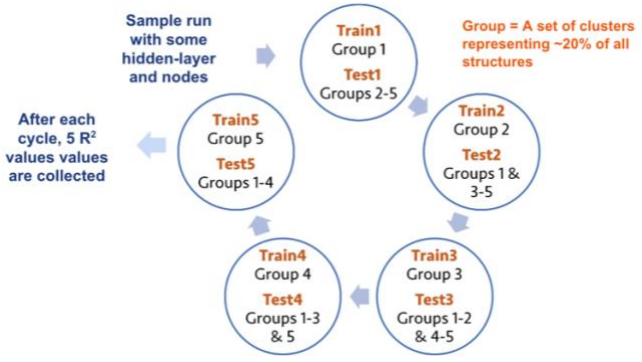


Figure 6. The representation of the train/test cycle that results in determining an R^2 value for each of the five assignments of the groups.

Finally, the effect of the NN architecture was tested by varying the value of hidden layers and nodes for each cycle. For this study, we examined three, four, and five hidden layers, each containing 5, 10, 15, 20, 25, 30, and 35 nodes.

4. RESULTS

Figures 7 and 8 plot the results of the random and clustered grouping of the training set. Each figure displays the average R^2 value — each collected from one complete round of the train/test cycle — for each hidden layer/node combination. The 20%/80% sized training sets are indicated by the blue and red lines, respectively.

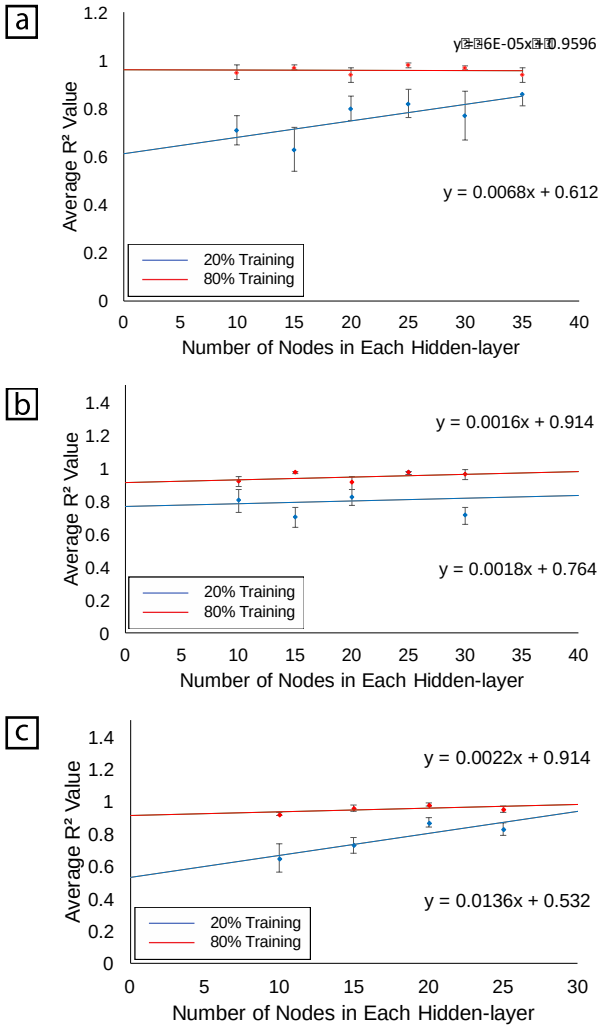


Figure 7. Three linear plots depicting averaged R^2 values and their uncertainties, where the NN is trained with three (in a), four (in b), and five (in c) hidden layers. The red line shows the 80% training set, and the blue line shows the 20% set.

Figures 7a, 7b, 7c, are the analysis for the randomly arranged groups. Figure 7a has three hidden layers, Figure 7b has four hidden layers, and Figure 7c has five hidden layers. Each figure displays averaged R^2 values as the number of nodes increased. The 80% training sets in Figures 7a, 7b, and 7c, have much smaller slopes and higher y-intercepts compared to the clustered data. Conversely, the 20% training sets have greater slopes and lower y-intercepts. There is less uncertainty in each of the averaged R^2 values in the 80% sets than in the 20% sets, as demonstrated by the smaller error bars. In general, the 80% training sets have higher R^2 values compared to the 20% training sets.

Figures 8a, 8b, 8c, show the results for the clustered groups. Figure 8a has three hidden layers, Figure 8b has four hidden layers, and Figure 8c has five hidden layers. In general, the 80% training sets in the clustered data have greater slopes and higher y-intercepts. The 20% training sets have smaller slopes and lower y-intercepts. The uncertainties of the 80% and 20% training sets appear to be similar. The 80% training sets have higher R^2 values, while the 20% training sets have lower R^2 values. In both instances (regarding random and clustered populations), the training set with more molecules resulted in an increase in R^2 values. These data are summarized in Table 1.

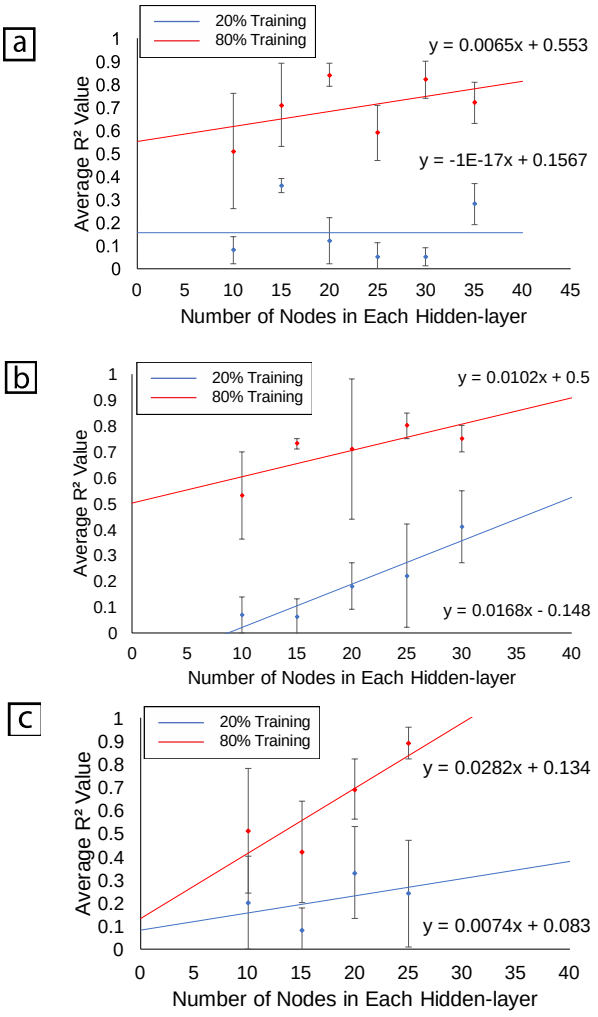


Figure 8. Three linear plots depicting averaged R^2 values and their uncertainties where the NN is trained with three (in a), four (in b), and five (in c) hidden layers. The red line shows the 80% training set, and the blue line shows the 20% set.

5. DISCUSSION

In the random data, the slopes for the 20% training sets are larger compared to the slopes for the 80% training sets. Even though the 20% trained data have much lower y-intercepts (meaning that they are not as accurate as the 80% data for small number of nodes), the slopes mean they eventually make up the difference as the number of nodes increases. One reason for the small slopes in the 80% data is that the R^2 values are already close to the maximum value of 1. This indicates that with a large amount of data, this NN is able to reliably calculate the DFTB+ energy for structures that are similar to those in its training set, regardless of the NN architecture. If a limited amount of data is available for the training set, reliable predictions can still be made with a NN, so long as the NN architecture is large.

This is different for the clustered data. The slopes (in general) for the 20% trained data are much smaller than the slopes for the 80% trained data. This means that smaller training data sets will require a very large number of nodes if they are going to catch up to the 80% data, if at all. Furthermore, contrary to the random data, for the larger training sets, there appears to be a much stronger correlation between NN architecture and the number of nodes in each hidden layer. This probably stems from the NN needing more extensive architectures to help predict structures that are not in the training set.

Table 1: Summary of the slopes and y-intercepts from Figures 7 and 8.

	20% Training		80% Training	
	Slope	y-intercept	Slope	y-intercept
Random Grouping				
3 Hidden Layers	0.0068	0.612	-6.00E-05	0.9596
4 Hidden Layers	0.0018	0.764	0.0016	0.914
5 Hidden Layers	0.0136	0.532	0.0022	0.914
Clustered Grouping				
3 Hidden Layers	-1.00E-17	0.1567	0.0065	0.553
4 Hidden Layers	0.0168	-0.148	0.0102	0.5
5 Hidden Layers	0.0074	0.083	0.0282	0.134

The larger y-intercepts for the 80% cases shows that they are much better at predicting structures outside of the training set for small NN architectures. It follows that when more examples are shown to the NN, it translates to better accuracy. The fact that the y-intercept drops significantly between the random and clustered data suggests that it matters to train the NN in a specific way if it needs to accommodate structures that it has never seen before. This is due to the fact that the random training set is comprised of all types of structures, while the clustered case attempts to segregate molecule types. and this is maintained in forming the training and testing groups. As a result, a false sense of accuracy exists for structures that have never been seen before when training with a random starting population.

Neural network architectures and training set size are important when trying to apply the NN calculator to structures that were not in the training set. This is illustrated by comparing the random training sets to the clustered ones. For the random training sets, regardless of the NN arrangement, it appears that we get at least a reasonable, if not very close, match to the DFTB+ energy value. Conversely, in the clustered data, we get a dramatically-reduced ability to predict the energy of the testing set structures, especially for architectures with a smaller number of hidden layers. This indicates that clustering should be used as a strategy to train NNs when it is expected that the NN will predict structures that are unique from the training set.

There are other examples of effective pairing of clustering data with other machine learning techniques. One study showed that the GA efficiently located the global minimum because it took advantage of the clustered configuration space [1]. Because of their potential for locating the global minimum, clusters and the GA are often found working simultaneously in areas of computational research. Another example showing the effectiveness of combining the GA

and clusters is found in a study where an approach known as the distributed hierarchical genetic algorithm (DHGA) was used for optimization, pattern matching, and space search exploration [4]. One study showed that when starting populations were clustered, the chances of finding the local minimum increased within a margin of error opposed to populations that were not subjected to the clustered approach [9]. The combination of the GA and clusters is constructive due to the ability of the GA to recognize a pattern when searching through the clustered configuration space. The GA can act upon this information and continue the search for the local minimum [1].

This project shows how clustering can work to illustrate how robust a trained NN is at accurately calculating properties of structures not in its training set. Traditional training of NN's with randomly chosen structures might provide a false sense of accuracy. The common starting conditions for NN are typically based on a random set, such as those found in the neural network potential (NNP) simulations that aimed to learn transferable potentials for organic molecules [15]. We have shown that certain NN architectures do not appear to do well when given a previously unseen structure. Clustering should be considered to determine a NN architecture that is robust enough to recognize structures that it has not seen before, since it can be quantified how accurate the NN is as a part of the training/testing sets. When randomly selected training/testing sets are used, the NN will perform well, but that is expected, since the test set is composed of structures similar to those on which it was trained.

We believe clusters should help improve the search scheme when deciding to create improved sophisticated programs of chemically relevant molecules in the potential energy surface. By learning to make the NN more robust in this project, we can discuss training a NN to generate starting populations for GAs. We envision a strategy where clustering is used to group together previous search results, and new starting populations can be quickly evaluated using a NN and compared to the clustered data to find new structural motifs and start new GA searches in unseen areas of the potential energy surface. This work informs how the NN should be constructed for this eventual application of a thorough and complete search for the global minimum. Ultimately, we have showed how training a NN requires some analysis to determine an architecture that will be robust enough to predict results from trial cases that it has not seen before using clusters.

6. CONCLUSION

In this paper, we demonstrate the importance of training NNs for calculating the energy of molecular structures using clustering to group together the training and testing data if it is expected that the NN will evaluate new structures that are outside of the training set. Using a random grouping in the training and testing sets resulted in the NN being able to almost exactly reproduce the correct energies of all tested structures. However, when the NN was trained with a group of clustered structures and tested on a separate group of clustered structures, the NN performed much worse. Enlarging the training set and increasing the number of hidden layers and nodes dramatically improves the ability of the NN to predict molecular energies of structures that are not similar to those in the training set. This approach provides a framework to determine the proper architecture to train more robust NNs with a quantifiable metric.

7. REFLECTION

I (CB) am excited to have been a part of the Blue Waters Student Internship Program. The financial support replaced having to work a part-time job, which provided the time necessary to complete the work for this publication. Furthermore, the two-week long computational science institute at the University of Illinois at Urbana-Champaign, hosted by the Shodor foundation, was a great experience. I really enjoyed learning about the many applications of high-performance computing, and the whole experience allowed me to build upon my computer science and chemistry backgrounds. This whole process really opened my mind to the opportunities in this field, and I hope to pursue these opportunities in the future. I would like to thank everyone involved with this program for a genuinely great experience.

8. REFERENCES

[1] Ulrich Bodenhofer. 2004. *Genetic Algorithms: Theory and Applications*. Fuzzy Logic Laboratorium Linz-Hagenberg. Johannes Kepler University, Linz, Austria. Retrieved from https://www.flll.jku.at/div/teaching/Ga/GA-Notes.pdf

[2] Yudong Cao, Gian G. Guerreschi, and Alán Aspuru-Guzik. 2017. Quantum Neuron: An elementary building block for machine learning on quantum computers. arXiv: 1711.11240. Retrieved from https://arxiv.org/abs/1711.11240

[3] Ye Chen, Meng Liu, Jianhua Chen, Yuqiong Li, Cuihua Zhao, and Xiao Mu. 2018. A density functional based tight binding (DFTB+) study on the sulfidization-amine flotation mechanism of smithsonite. *Appl. Surf. Sci.* 458 (Nov. 2018), 454-463. DOI: https://doi.org/10.1016/j.apsusc.2018.07.014

[4] Gautam Garai and B.B. Chaudhuri. 2007. A distributed hierarchical genetic algorithm for efficient optimization and pattern matching. *Pattern Recogn.* 40, 1 (Jan. 2007), 212-228. DOI: https://doi.org/10.1016/j.patcog.2006.04.023

[5] Daniel George and E.A. Huerta. 2018. Deep Learning for real-time gravitational wave detection and parameter estimation: Results with Advanced LIGO data. *Phys. Lett. B.* 778 (Mar. 2018), 64-70. DOI: https://doi.org/10.1016/j.physletb.2017.12.053

[6] Florian Häse, Christoph Kreisbeck, and Alán Aspuru-Guzik. 2017. Machine learning for quantum dynamics: deep learning of excitation energy transfer properties. *Chem. Sci.* 8, 12 (Dec. 2017), 8419-8426. DOI: https://doi.org/10.1039/c7sc03542j

[7] Anil K. Jain. 2010. Data clustering: 50 years beyond K-means. *Pattern Recogn. Lett.* 31, 8 (Jun. 2010), 651–666. DOI: https://doi.org/10.1016/j.patrec.2009.09.011

[8] Mathias S. Jørgensen, Michael N. Groves, and Bjørk Hammer. 2017. Combining Evolutionary Algorithms with Clustering toward Rational Global Structure Optimization at the Atomic Scale. *J. Chem. Theory Comput.* 13, 3 (Feb. 2017), 1486-1493. DOI: https://doi.org/10.1021/acs.jctc.6b01119

[9] Nicholas Kellas and Michael N. Groves. 2020. Improvement of the Evolutionary Algorithm on the Atomic Simulation Environment Though Intuitive Starting Population Creation and Clustering. *Journal of Computational Science Education* 11, 2 (Apr. 2020), 29-35. DOI: https://doi.org/10.22369/issn.2153-4136/11/2/5

[10] Alireza Khorshidi and Andrew A. Peterson. 2016. Amp: A modular approach to machine learning in atomistic simulations. *Comput. Phys. Commun.* 207, (Oct. 2016), 310-324. DOI: https://doi.org/10.1016/j.cpc.2016.05.010

[11] Philipp Koehn. 1994. *Combining Genetic Algorithms and Neural Networks: The Encoding Problem*. Master's thesis. The University of Tennessee, Knoxville, Knoxville, TN.

[12] Esben L. Kolsbjerg, Andrew A. Peterson, and Bjørk Hammer. 2018. Neural-network-enhanced evolutionary algorithm applied to supported metal nanoparticles. *Phys. Rev. B.* 97, 19 (May 2018), 195424. DOI: https://doi.org/10.1103/PhysRevB.97.195424

[13] Ask Hjorth Larsen, et al. 2017. The atomic simulation environment—a Python library for working with atoms. *J. Phys.-Condens. Mat.* 29, 27, (Jun. 2017), 273002. DOI: https://doi.org/10.1088/1361-648X/aa680e

[14] Marwin H. S. Segler, Thierry Kogej, Christian Tyrchan, and Mark P. Waller. 2018. Generating Focused Molecule Libraries for Drug Discovery with Recurrent Neural Networks. *ACS Cent. Sci.* 4, 1 (Dec. 2017), 120-131. DOI: https://doi.org/10.1021/acscentsci.7b00512

[15] Justin S. Smith, Olexandr Isayev, and Adrian E. Roitberg. 2017. ANI-1: an extensible neural network potential with DFT accuracy at force field computational cost. *Chem. Sci.* 8, 4 (Feb. 2017), 3192-3203. DOI: https://doi.org/10.1039/c6sc05720a

[16] Logan Ward and Chris Wolverton. 2017. Atomistic calculations and materials informatics: A review. *Curr. Opin. Solid St. M.* 21, 3 (Jun. 2017), 167-176. DOI: https://doi.org/10.1016/j.cossms.2016.07.002

[17] Yi Yang, Dong Xu, Feiping Nie, Shuicheng Yan, and Yueting Zhuang. 2010. Image Clustering Using Local Discriminant Models and Global Integration. *IEEE T. Image Process.* 19, 10 (Apr. 2010), 2761-2773. DOI: https://doi.org/10.1109/TIP.2010.2049235

[18] Xinjie Yu and Mitsuo Gen. 2010. *Introduction to Evolutionary Algorithms.* Springer-Verlag London. DOI: https://doi.org/10.1007/978-1-84996-129-5

Molecular Simulations for Understanding the Stabilization of Fullerenes in Water

Kendra Noneman
Micron School of Materials Science and Engineering
Boise State University
Boise, ID
kendranoneman@u.boisestate.edu

Christopher Muhich
School for Engineering of Matter, Transport, and Energy
Arizona State University
Tempe, AZ

Kevin Ausman
Department of Chemistry
Boise State University
Boise, ID

Mike Henry
Micron School of Materials Science and Engineering
Boise State University
Boise, ID

Eric Jankowski
Micron School of Materials Science and Engineering
Boise State University
Boise, ID
ericjankowski@boisestate.edu

ABSTRACT

Making materials out of buckminsterfullerene is challenging, because it requires first dispersing the molecules in a solvent, and then getting the molecules to assemble in the desired arrangements. In this computational work, we focus on the dispersion challenge: How can we conveniently solubilize buckminsterfullerene? Water is a desirable solvent because of its ubiquity and biocompatibility, but its polarity makes the dispersion of nonpolar fullerenes challenging. We perform molecular dynamics simulations of fullerenes in the presence of fullerene oxides in implicit water to elucidate the role of interactions (van der Waals and Coulombic) on the self-assembly and structure of these aqueous mixtures. Seven coarse-grained fullerene models are characterized over a range of temperatures and interaction strengths using HOOMD-Blue on high performance computing clusters. We find that dispersions of fullerenes stabilized by fullerene oxides are observable in models where the net attraction among fullerenes is about 1.5 times larger than the attractions between oxide molecules. We demonstrate that simplified models are sufficient for qualitatively modeling micellization of these fullerenes and provide an efficient starting point for investigating how structural details and phase behavior depend upon the inclusion of more detailed physics.

KEYWORDS

Self assembly, Computer simulation, GPUs, Molecular dynamics, Undergraduate research, Blue Waters

1 INTRODUCTION

Buckminsterfullerene (C_{60}) can be dispersed in water, starting from both pure solid and organic solutions, rendering this colloid the most environmentally relevant form of the fullerenes [1–7]. The ability to disperse C_{60} colloids in water, termed "$n$$C_{60}$", is of interest from a nanomanufacturing perspective, because such solutions offer a starting point for using thermodynamics to self-assemble nanostructures from these organic conducting building blocks without the need for volatile solvents. Early reports of C_{60} cytotoxicity were followed by substantively conflicting reports of the material's environmental and biological behavior [2, 8–17]. This controversy has elevated C_{60} to a status as a touchstone nanoparticle, in many ways serving as a proxy for the scientific community's responsible development of nanomaterials as an industry.

Since its first laboratory synthesis in 1994 [5], several aspects of $n$$C_{60}$'s structure, stability, and reactivity have defied explanation. Of the reported $n$$C_{60}$ laboratory synthesis methods, most involve transfer of the fullerene material from an organic solvent into water [1, 2, 6, 18–20]. One approach, however, known as AQU/$n$$C_{60}$, involves simple extended stirring in water for a period of days to months [6, 7, 19, 21, 22]. This last technique is notoriously inconsistent and unreliable, despite it being the most environmentally-relevant approach. Most critically, the surface chemistry of the colloidal particles must be charged or hydrophilic, or both, in order to render the particles water-stable. The most significant breakthrough in understanding the nature of this surface chemistry came in 2012 with the observation that for AQU/$n$$C_{60}$ this hydrophilic chemistry results from an unusually-stable epoxide derivative of C_{60}, which is formed through reaction of trace levels of atmospheric ozone with buckminsterfullerene [23]. This same $C_{60}O$ derivative has been shown, at least in some samples, to stabilize aqueous colloidal aggregates of C_{60} ($n$$C_{60}$) in three of the main organic solvent-based synthesis techniques: the exchange methods of hexane/isopropyl alcohol/water (HIPA), tetrahydrofuran/water (THF), and toluene/tetrahydrofuran/acetone/water (TTA) [20]. The present work aims to elucidate the self-assembly of fullerenes into colloidal particles under conditions where a mixture of C_{60} and $C_{60}O$ are present. This initial work explores equimolar quantities of each, aiming to understand the minimal physics required to describe colloidal stabilization of C_{60} and $C_{60}O$ in water. What C_{60} and

https://doi.org/10.22369/issn.2153-4136/12/1/6

$C_{60}O$ characteristics and conditions are sufficient for $C_{60}O$ to stabilize micelles (Figure 1) of C_{60}? Are there other thermodynamically stable arrangements of C_{60} and $C_{60}O$?

We approach these questions of fullerene stability through the lens of molecular simulations and micelle self-assembly. Molecular dynamics (MD) and Monte Carlo (MC) simulations have been employed to investigate how the properties of amphiphilic molecules can self-assemble into higher-ordered structures including micelles [24–27]. These simulations provide a convenient way to study how the thermodynamic stability of micelles and bilayers depends on interactions and geometric constraints, as predicted by Israelachvili, Mitchell, and Ninham [28]. For example, MD simulations of model surfactants in a solvent show that micelles of varying structure (disk, cylindrical, and spherical) can readily self-assemble over simulation timescales [26]. More recent work using a more complex MARTINI model of surfactants in water demonstrates that the long timescales of lipid bilayer transformations can be accessed in MD simulations [25]. Representing solvent molecules implicitly using Langevin or Brownian dynamics is another technique for accessing longer timescales, and the work of Noguchi et al. demonstrates that micelles can self-assemble from amphiphilic molecules and aggregate into vesicles in the right conditions [24]. The thermodynamic stability of spherical micelles, cylindrical micelles, vesicles, and bilayers was demonstrated to be a function of amphiphile shape (here, dumbbell size ratio) in MC simulations by Avvisati et al. [27]. In summary, simplified models of amphiphiles have been successfully deployed to study thermodynamic phase behavior and self-assembly for a wide variety of amphiphilic molecules. Here, our coarse model of amphiphilic $C_{60}O$ is developed in the spirit of Refs. [27] and [24] and we investigate its self assembly with C_{60} in an implicit solvent with MD simulations.

2 MODEL

We use MD simulations to investigate a sequence of seven coarse-grained models of C_{60} and $C_{60}O$ to determine how our choice of model determines the structures that self-assemble. In all of the models, C_{60} is represented by a single coarse-grained simulation element (Figure 2(h)), $C_{60}O$ is represented by two spherical simulation elements (Figure 2(a–g)), and water is modeled implicitly using Langevin dynamics [24, 29]. The hydrodynamic drag on each spherical simulation element is described by the Stokes formula

$$F_d = 3\pi\eta d\vec{v} \tag{1}$$

where η is the viscosity of the fluid, d is the particle diameter, and $\vec{v}$ is the particle velocity. Random forces on each spherical simulation element from the implicit solvent are described by $\zeta(t)$, which is related to the drag force through the fluctuation dissipation theorem

$$\left\langle \zeta(t)^2 \right\rangle = 2F_d T \tag{2}$$

and where $\langle \zeta(t)\zeta(t') \rangle = 0$. The net effect of modeling the solvent implicitly through Langevin dynamics is that the simulation elements are thermostatted not to travel so fast as to be numerically unstable, but with dynamics and computational cost sufficient to reach equilibration in minutes to hours. We also note that the inertial terms (masses, moments of inertia) do not factor into the equilibrium structures sampled.

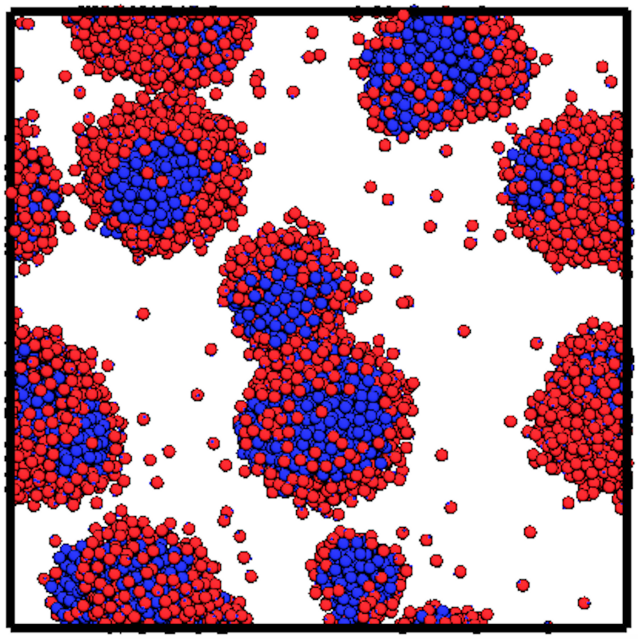


Figure 1: Representative snapshot of C_{60} (blue spheres) stabilized in water with $C_{60}O$ (red spheres with small blue dots) acting as a surfactant between the nonpolar C_{60} and polar water.

The seven models we consider here are chosen to answer: *What combinations of van der Waals interaction strengths, partial charges, and molecular geometry are sufficient to predict the formation of C_{60} micelles stabilized by $C_{60}O$?* The models also roughly represent a gradient of increasing complexity, from simplified models that do not explicitly include long-range electrostatics, to models that have partial charges and moments of inertia informed by first-principles calculations.

(1) The geometry of $C_{60}O$ in Model 1 (Figure 2a) is designed using approximate atomic sizes and positions from the optimized OPLS-AA force field (optimized potentials for the simulation of liquids, all atom) [30].
(2) The geometry of $C_{60}O$ in Model 2 (Figure 2b) is informed by density-functional theory (DFT) calculations described later in this paper.
(3) Model 3 (Figure 2c) uses the same geometry as Model 2, but adds naïve partial charges to the coarse-grained C_{60} and O beads, creating a static dipole across the $C_{60}O$ molecule. Model 3 also scales down the energies of the O-O and O-C_{60} interactions.
(4) Model 4 (Figure 2d) improves upon Model 3 by applying partial charges calculated from first principles and using interaction energies scaled to the represent accurate van der Waals forces.
(5) Model 5 (Figure 2e) is identical to Model 4, except the charge magnitudes are scaled down so the dipole moment on $C_{60}O$ has the correct magnitude, in exchange for lower partial charges.

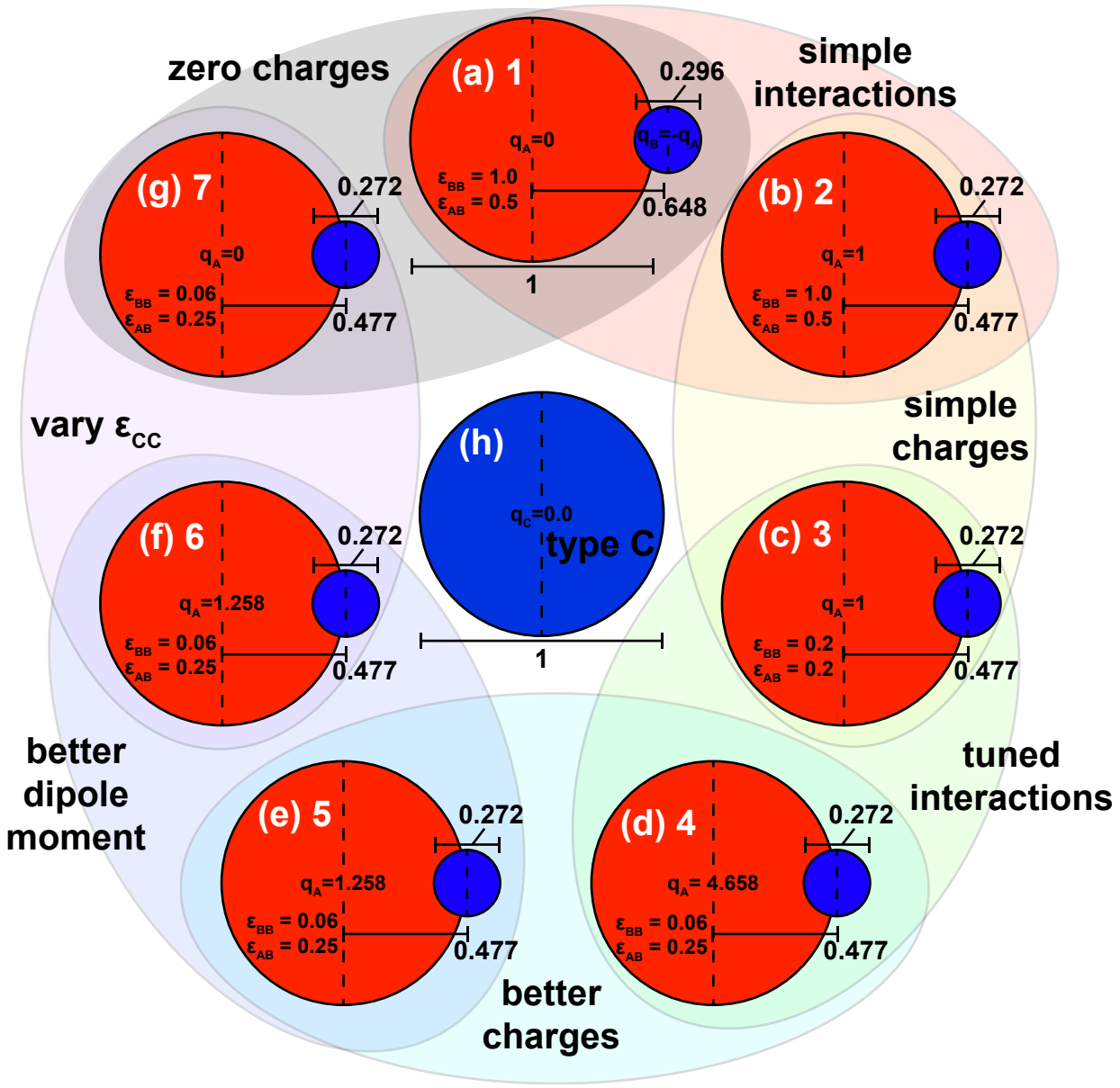


Figure 2: Seven models of $C_{60}O$ are tested for their self-assembly properties with C_{60} (center) in implicit solvent. Clockwise from the top, each successive model is a perturbation to the prior one: Model 2 updates the location and size of the O-group and adds naïve charges to Model 1. Model 3 has more accurate Lennard-Jones (LJ) interaction energies. Model 4 has partial charges informed by density-functionary theory (DFT) calculations and LJ interactions drawn from literature. Model 5 scales the charge magnitudes of Model 4 to have an accurate dipole moment. In Model 6 we keep the Model 5 C_{60} details, but vary ε_{CC}. Model 7 is identical to Model 6 (ε_{CC} varied), but omits partial charges.

(6) Model 6 (Figure 2f) takes a different view and explores how the C_{60}-C_{60} interactions influence phase behavior (by varying ε_{CC}), using the same $C_{60}O$ model as Model 5.
(7) Model 7 (Figure 2g) is identical to Model 6, except it omits the partial charges on $C_{60}O$.

In concert, these models comprise a sequence of assumptions about how C_{60} and $C_{60}O$ behave that begins simply, adds in electrostatics, more accurate van der Waals forces, more accurate charges, and dipoles that enable systematic evaluation of which features are necessary for capturing hypothesized C_{60}-$C_{60}O$ phase behavior.

The potential energy between a pair of simulation elements is described by the Lennard-Jones interaction potential

$$U_{LJ}(r_{ij}) = 4\varepsilon_{kl}\left(\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right), \quad (3)$$

where r_{ij} is the center-to-center distance between particles i and j, ε_{kl} is the Lennard-Jones well depth, σ_{kl} is the Lennard-Jones length scale, where k and l indicate particle types, with $k \in \{A, B, C\}$ and $l \in \{A, B, C\}$ in this work. We use Lorentz-Berthelot mixing rules where $\sigma_{kl} = \frac{\sigma_{kk}+\sigma_{ll}}{2}$ represents the sum of the radii of particles i and j whose types are k and l, respectively, and cross-interactions of energy scales are ($\varepsilon_{kl} = \sqrt{\varepsilon_{kk}\varepsilon_{ll}}$) [31, 32]. Equation 3 has a global minimum at $U_{LJ}(r_{ij} = \sigma_{kl}2^{1/6}) = \varepsilon_{kl}$, so we may select ε_{kl} to represent qualitative attractions between model elements A (C_{60} in $C_{60}O$, red elements in Figure 2), B (O, smaller blue elements in Figure 2), and C (C_{60}, Figure 2h).

To minimize confusion that arises when mentally converting between "particle type", "atom types", and "molecules", we henceforth adopt the following naming conventions:

- "A", "B", and "C" are used to refer to the spherical simulation elements that are used to models of $C_{60}O$ and C_{60}, and will only be used in the context of parameters describing interactions between simulation elements (e.g. ε_{BB} and q_A in Table 1) and structural metrics such as radial distribution functions between particles of type B g_{BB}.
- The symbols $C_{60}O$ and C_{60} are used to refer to molecules of $C_{60}O$ and C_{60} generically, both within and outside the context of the models presented here.
- In structural analysis, "A" and $C_{60}O$ are interchangeable, and "C" and C_{60} are interchangeable. That is, a shortcut heuristic for this work is that "A" = $C_{60}O$ and "C" = C_{60}, and we include both sets of symbols as a reminder of the distinction between model implementation ("A", "B", and "C") and molecules (C_{60} and $C_{60}O$).

To minimize unit conversion errors and maximize the interpretability of predictions with other simulation results, we employ base units of mass $M = 1.20 \times 10^{-24}$ kg, length $\sigma = 1.01 \times 10^{-9}$ m, and energy $\varepsilon = 3.24 \times 10^{-20}$ J to convert between physical fullerene parameters and near-unity dimensionless simulation parameters. That is, when we report $\varepsilon_{AB} = 0.5$, it means that $\varepsilon_{AB} = 0.5\varepsilon = 1.62 \times 10{-}20$ J. Using dimensionless simulation parameters allows for quick inspection of relative magnitudes and aids with analysis. The base units of mass M and length σ correspond to the mass and van der Waals diameter [30] of C_{60}, respectively. The base unit of energy ε corresponds to the minimum potential energy of a C_{60} dimer in water and is within the range of calculations from prior work (2.7×10^{-20} to 3.7×10^{-20}) [33, 34]. Derived units of time $\tau = \sqrt{\frac{M\sigma^2}{\varepsilon}}$ are 6.15×10^{-12} s. One quirk of dimensionless simulation units is the convention of reporting temperatures T in terms of dimensionless energy, which can be converted to Kelvin using $T_{Kelvin} = \frac{\varepsilon T}{k_B}$, where $k_B = 1.38 \times 10^{-23}$ J/K. With $\varepsilon = 3.24 \times 10^{-20}$ J , $T = 1$ corresponds to 2348 K, but we caution overinterpretation of this absolute temperature: Rather, we are reminded that constructing a model in terms of ε—a coarse-grained aqueous C_{60} interaction, whatever it may be—we are able to investigate what other model parameters are sufficient for observing micellization. Framed this way, correspondence of room-temperature (298 K) micellization in experiments with the present model's predictions at $T = 0.7$ would provide rationale for re-interpreting $\varepsilon = \frac{298\text{ K}\times1.38\times10^{-23}\text{ J/K}}{0.7} = 5.87 \times 10^{-21}$ J within the context of this model.

Table 1: Interaction parameters representing oxygen-oxygen ε_{BB}, functionalized C_{60}-oxygen ε_{AB}, and unfunctionalized C_{60}-C_{60} attractions, as well as partial charges on functionalized C_{60} (q_A) and O (q_B) groups.

Model	ε_{BB}	ε_{AB}	ε_{CC}	q_A	q_B
1	1	0.5	1	0	0
2	1	0.5	1	1	-1
3	0.2	0.2	1	1	-1
4	0.06	0.25	1	4.658	-4.658
5	0.06	0.25	1	1.257	-1.257
6	0.06	0.25	1.0 to 2.0	1.257	-1.257
7	0.06	0.25	1.0 to 2.0	0	0

The $C_{60}O$ are represented as rigid bodies [35]. This minimal model allows for the shapes of these molecules to be realistically represented without having to keep track of their computationally expensive internal degrees of freedom. We choose interaction strengths (ε_{AA}, ε_{BB}, and ε_{CC}) in terms of ε to systematically model systems with varying degrees of attraction. In this work, we focus on understanding how the choice of ε_{kl}'s determines the structures that emerge (Table 1).

2.1 Partial Charge Calculation

To determine charge distribution on the fullerene and the position of the oxide oxygen (Models 4, 5, 6, and 7), we employ quantum chemical calculations using density-functional theory (DFT). The calculations are carried out using Quantum Espresso [36] with periodic boundary conditions using the Perdew-Burke-Ernzerhof (PBE) generalized gradient approximation (GGA) exchange-correlation functional [37]. Projector augmented wave (PAW) pseudopotentials [38] enable explicit calculation of the 2s and 2p electrons of C and O. Calculations are conducted at the Γ point. C_{60} is initialized in large cube with edges greater than 20Å (roughly three times the diameter of the fullerene), so as to minimize periodic image interactions.

The wave functions are composed of summations of plane waves with energies up to 45 Rydberg (612 eV). No differences in geometry or energies were seen between the calculation using a 45 Ry cutoff energy and a calculation using a 40 Ry cutoff energy, suggesting that both cutoff energies are sufficient to accurately model the wave function. As both wave functions are calculated to ensure convergence, we used the more accurate 45 Ry wave function for all further analysis. Bader analysis [39] is used to determine the charge on the atoms in the Buckyball. The code used to calculate the Bader charges was written by the Henkelman group [40, 41].

3 METHODS

Coarse-grained molecular dynamics simulations are performed using the HOOMD-Blue simulation engine [42] on NVIDIA K20 GPUs

on the NCSA Blue Waters and Boise State Kestrel computing clusters. Each simulation is performed using a single K20 GPU, during which the potential energy and simulation snapshots of the particles are logged. Simulations are performed in the canonical (constant number of particles N, volume V, and temperature T) ensemble using the Nosé-Hoover thermostat [43] and the MTK equations [44, 45]. Quaternions are used to represent the orientational degrees of freedom of the $C_{60}O$ molecules. Long-range electrostatics are calculated using the particle-particle particle-mesh (PPPM) method [46]. Positions and quaternions are updated via the velocity Verlet integration of Newton's equations of motion after every dimensionless time step, $dt = 0.001\tau$ [47].

Simulation volumes are initialized on a square lattice with constant number density $\eta = 0.074$ and equal amounts of C_{60} and $C_{60}O$ in a cubic volume. Small debugging jobs are performed with 150 each of C_{60} and $C_{60}O$, and the largest jobs are performed with 8,000 of each. Simulation snapshots presented in the figures here are from production jobs with 3,375 each of C_{60} and $C_{60}O$, for 10,125 total simulation elements. We consider systems to be equilibrated once the potential energy reaches a stable average, which for the coldest systems studied here occurs after about 2×10^7 time steps in the largest system sizes, and in under 4×10^6 steps for production runs (Figure 3). After equilibration is achieved, we average statistically-independent samples (over 20 for all systems studied here), requiring 1×10^7 steps for the largest systems and 1×10^6 for the production systems. Simulation snapshots are visualized using Visual Molecular Dynamics (VMD) [48], and plotting of simulation data (potential energy trajectories and radial distribution functions $g(r)$) are performed with matplotlib [49].

We use $g_{i,j}(r)$ to measure local spatial correlations between particle types i and j and to distinguish between "dispersed", "aggregate-onset", "fluid clusters", "solid clusters", and "micelle" structures (Figure 4 and Figure 5). The key $g(r)$ metrics we focus on here are peaks corresponding to first- ($r = 1.15\sigma$) and second-nearest neighbor ($1.85\sigma < r < 2.3\sigma$) distances, and whether $g(r)$ converges to the system density at large separations ($\lim_{r\to 6} g(r) = 1$). The magnitude of the first-nearest neighbor peak is used to indicate relative contact frequencies, and a peak at $r = 1.15\sigma$ is expected, because the global minimum of Equation 3 occurs at $r = 1.12\sigma$, and the C_{60} simulation elements have the small oxide component that contributes slightly to average separations. The second-nearest neighbor peak is used to distinguish fluid clusters from dispersed systems. If $\lim_{r\to 6} g(r) > 1$, we expect to find tightly-packed clusters that do not uniformly fill the simulation volume. In micelle cases we use $g_{AC}(r)$ to quantify how "tightly" the shell covers the core of the micelle.

4 RESULTS AND DISCUSSION

We perform MD simulations of equimolar C_{60} and $C_{60}O$ in implicit solvent for seven models. For each model, we vary temperature T to observe transitions between phases. In Models 6 and 7, we vary temperature T and C_{60}-C_{60} interactions (ε_{CC}). After equilibration, simulation snapshots and radial distribution functions $g(r)$ are generated from representative microstates. In this section we present, interpret, and compare the transitions and structures for the seven

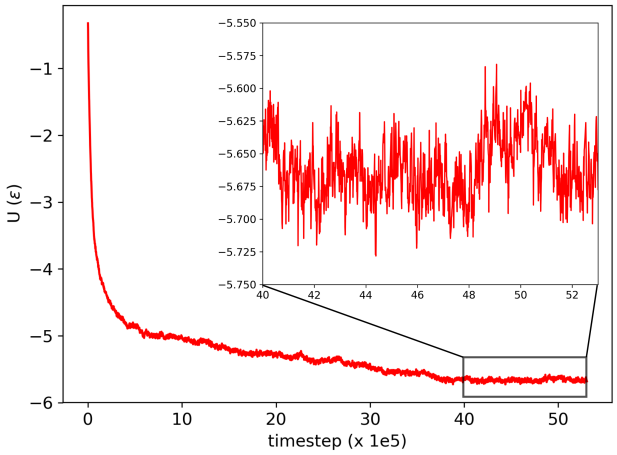


Figure 3: Per-molecule potential energy trajectory for a representative slow-to-relax simulation (Model 7, $T = 0.7$ $\varepsilon_{CC} = 1.5$) that reaches equilibration after 4 million steps and is then sampled fluctuating about a stable average for 1.3 million steps. Inset: Detail of equilibrium potential energy fluctuations.

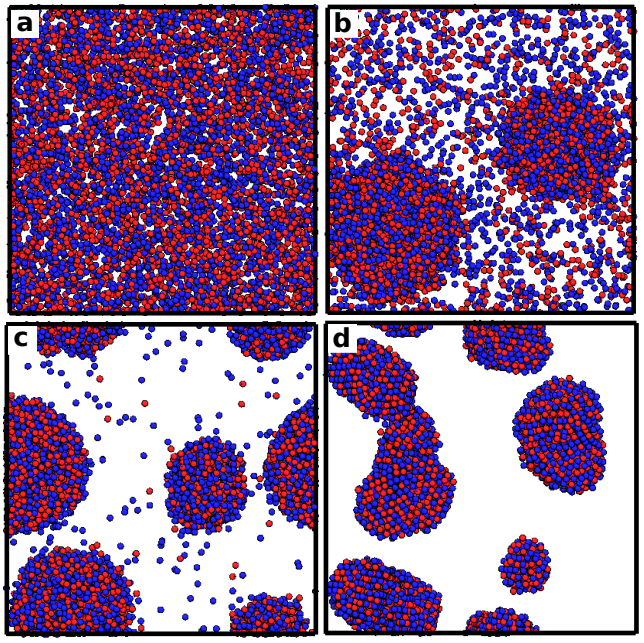


Figure 4: We observe that all seven models transition between the following four phases as temperature is decreased: (a) Dispersed: Molecules move freely and aggregate minimally, (b) Aggregate-onset: Formation of at least one region of relatively high molecular density, (c) Fluid clusters: Few un-aggregated molecules, but molecules still move freely within the clusters, and (d) Solid-like clusters: Rigid clusters where molecules vibrate about static positions within the aggregate.

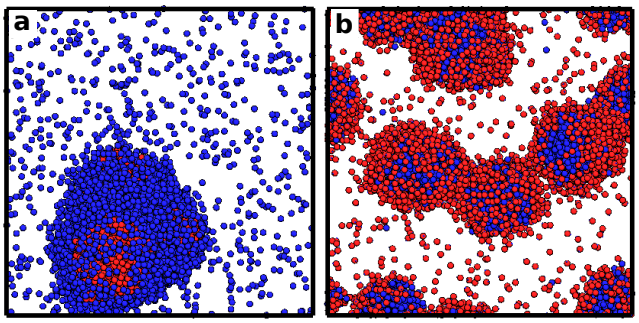


Figure 5: The "micelle" structures observed here are characterized by thin layers of one type of molecule encapsulating spherical clusters of the other molecule type. Here we observe both (a) micelles in which C_{60} encapsulates $C_{60}O$ (Model 4, $T = 0.7$), and (b) micelles in which $C_{60}O$ encapsulates C_{60} (Model 7, $T = 0.7$). Micelle stability is not observed for all models. When micelles are observed, it is at temperatures below aggregate-onset and only for sufficiently high attraction between C_{60} molecules (ε_{CC}).

models. The results presented here are for the production simulations of 10,125 simulation elements representing 3,375 C_{60} and 3,375 $C_{60}O$ molecules. Overall, we observe the same sequence of structural transitions as we decrease temperature: from dispersed, to aggregate-onset, then fluid, and finally solid-like (Figure 4). For a subset of models, micelles are observed to self-assemble below the aggregate-onset temperature.

(1) In Model 1, fluid clusters emerge between $T = 1.3$ and $T = 1.2$. Aggregation increases (as measured by $g(r = 1.15)$) for all simulation element types (AA followed by AC followed by CC) as temperature is lowered from $T = 1.2$ to $T = 0.7$. A crystallization transition is observed at $T = 0.6$.
(2) Model 2 adds charges relative to Model 1, and we observe the same behaviors as in Model 1 but with fewer, larger clusters (Figure 6b).
(3) Decreasing ε_{BB} in Model 3 causes phase transitions to occur at lower temperatures ($T = 0.9$ for fluid clusters, $T = 0.4$ for solid clusters), but the observed structures are the same as in Models 1 and 2. This is expected, because the thermodynamic driver for oxygen aggregation is smaller ($\varepsilon_{BB} = 0.2$ vs $\varepsilon_{BB} = 1.0$).
(4) Observations of Model 4 differ significantly from Models 1–3, and the model differs in the utilization of partial charges informed by DFT calculations and LJ interactions ($q_A = 4.658$ and $\varepsilon_{AB} = 0.25$). Stronger $C_{60}O$ fullerene cage (AA) aggregation is observed at $T = 1.8$, and shells of C_{60} form around $C_{60}O$ aggregates around $T = 0.9$ (inversions of the experimentally hypothesized micelles, Figure 5(a)). Crystallization occurs at $T = 0.7$.
(5) In Model 5, charge magnitudes are decreased to model a more accurate dipole moment, but the εs from Model 4 are retained. We observe fluid clusters at $T = 0.9$ and solid clusters at $T = 0.5$, but overall observations follow models 1–3: mixed fluid and solid clusters.

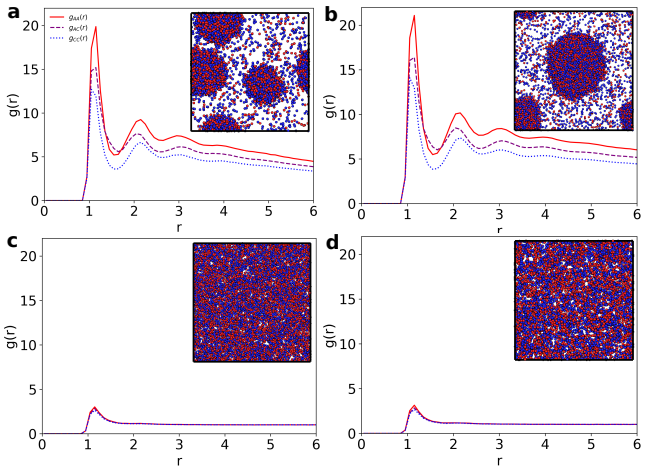


Figure 6: Measurements of spatial correlations of simulation elements and representative snapshots at $T = 1.0$ for a) Model 1, b) Model 2, c) Model 3, and d) Model 5. At this temperature, aggregation is observed in Models 1 and 2 but not Models 3 and 5. Red g_{AA} measures correlations between the fullerene cages of $C_{60}O$, which are rendered as red spheres. Dashed purple g_{AC} measures correlations between C_{60}'s and the fullerene cages of $C_{60}O$. Dotted blue g_{CC} measures correlations between C_{60}, which are rendered as blue spheres.

(6) Within Model 6 we vary ε_{CC} from 1.0 to 2.0 over $0.6 \leq T \leq 1.0$. Higher values of ε_{CC} promote C_{60}-C_{60} aggregation. Within these ranges of T and ε_{CC} we observe micelles of $C_{60}O$ around C_{60}. (Fig. 5(b))
(7) Within Model 7 we also vary ε_{CC} from 1.0 to 2.0 over $0.6 \leq T \leq 1.0$, but without any long-range electrostatics. Shells of $C_{60}O$ around C_{60} still begin to form, but the aggregates have slightly different structures without the charges.

In summary, Models 6 and 7 are sufficient to observe the experimentally hypothesized micelle stabilization of $C_{60}O$ monolayers forming around spherical clusters of C_{60} in solution. The only difference between these two models is the presence of electrostatic charges in Model 6 that are absent in Model 7. The self-correlations of $C_{60}O$'s and C_{60}'s are measured to be higher in Model 6 (Figure 9), so the presence of charges has measurable impact on local structural details. In both Models 6 and 7, $C_{60}O$ is observed to play the role of micelle surfactant around clusters of C_{60} when $\varepsilon_{CC} \geq 1.5$.

4.1 Models 1, 2, 3, and 5

The structures observed in Models 1, 2, 3, and 5 are very similar. At colder temperatures, $C_{60}O$'s and C_{60}'s aggregate, but they do not demonstrate thermodynamic preference for spatial segregation by type. Representative snapshots are presented in Figure 6. The major difference between Models 1 and 2 versus 3 and 5 is the temperature at which accumulation begins. Overall, these results indicate that changing charge magnitudes between 0 and 1.3 and varying the oxygen attraction parameter (ε_{BB}) aren't primary facors for micellization (recall Table 1).

4.2 Model 4

Structures observed in Model 4 differ significantly from Models 1-3 and 5. Model 4 has oxygen interactions informed by the OPLS and MARTINI force fields [30]. Because of this, the oxide molecules aggregate strongly at temperatures at and below $T = 1.8$. At temperatures around $T = 0.9$, nearly all of the $C_{60}O$'s have grouped up into one large cluster, but the C_{60}'s remain detached. As the temperature is lowered further, the C_{60}'s begin sticking to the surface of the $C_{60}O$ cluster and form a shell around the outside.

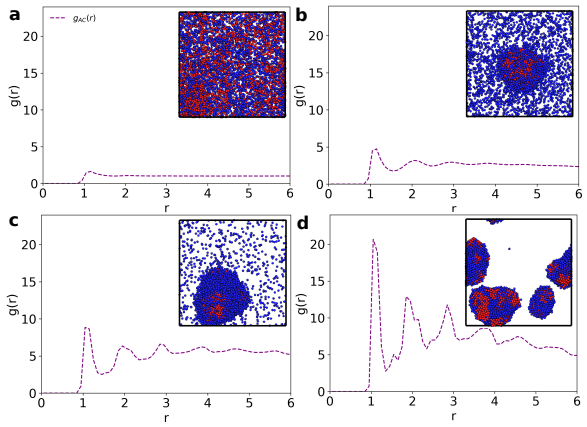


Figure 7: Local structure and representative snapshots from Model 4 at various temperatures. $g_{AA}(r)$ and $g_{CC}(r)$ have been excluded, as the extremely charged $C_{60}O$ molecules ($q_A = 4.658$) are considerably attractive and crystalline. a) $T = 1.8$ aggregation-onset, b) $T = 0.9$ $C_{60}O$'s aggregate into clump and C_{60}'s remain loose, c) $T = 0.7$ C_{60}'s form a shell around the crystalline $C_{60}O$ cluster, and d) $T = 0.4$ tightly packed clusters, but too cold for C_{60}'s and $C_{60}O$'s to reassemble.

4.3 Models 6 and 7

Experimentally-hypothesized micelles of C_{60} surrounded by $C_{60}O$ are observed to self-assemble in Models 6 and 7. These two models are used to explore the role ε_{CC} plays in micellization based on the interaction set of Model 5 with (Model 6) and without (Model 7) charges. Model 6 contains the exact long-range electrostatics of Model 5, while in Model 7 these charges are turned off. For both models, we perform simulations with temperatures ranging from $T = 0.6$ to $T = 1.0$ and ε_{CC} ranging from $\varepsilon_{CC} = 1.0$ to $\varepsilon_{CC} = 2.0$.

We observe significant differences in the structures that are thermodynamically assembled by Models 6 and 7:

(1) $C_{60}O$'s aggregate at higher temperatures and with greater frequency in Model 6 (charged).

(2) The first-nearest neighbor peak magnitudes ($g(r = 1.15)$) for Model 6 are greater than Model 7 at the same temperatures (Figure 8 and 9. The difference in peak magnitudes between $C_{60}O$-$C_{60}O$, $C_{60}O$-C_{60}, and C_{60}-C_{60} is also more spread out for the charged system (Figure 8 and 9). This signifies that charges increase the relative contact frequencies.

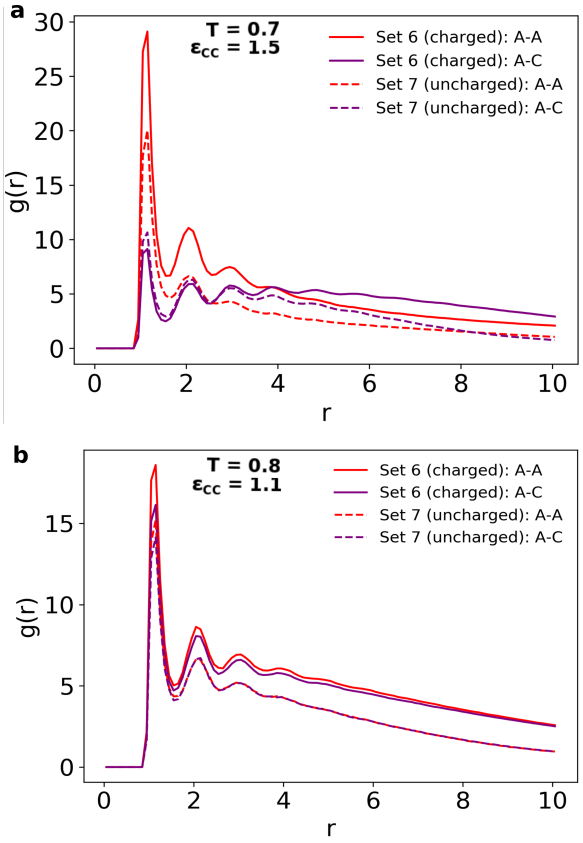


Figure 8: Comparisons of $g_{AA}(r)$ and $g_{AC}(r)$ for the charged Model 6 and uncharged Model 7 show greater oxide aggregation in the charged cases. a) $T = 0.7$ and $\varepsilon_{CC} = 1.5$, b) $T = 0.8$ and $\varepsilon_{CC} = 1.1$. The concave-down shape of g_{AC} in solid purple in (a) with a moving average trending up (in this case with a moving average maximum around $g_{AC}(r = 5) = 6$) before turning downwards is an unusual $g(r)$ feature that is representative of micelles with a sheath of either A or C around the other type.

(3) Lastly, Model 6 (with charged $C_{60}O$'s) consistently has a lower quantity of clusters compared to Model 7 (with uncharged $C_{60}O$'s).

We interpret these observations to indicate that the dipole-dipole interactions available in Model 6 facilitate the aggregation of $C_{60}O$ molecules and enhance the exclusion of C_{60} from $C_{60}O$. In sum, Models 6 and 7 demonstrate that stabilization of C_{60} clusters in water by $C_{60}O$ requires the effective attraction of C_{60} be roughly 50% greater than the net attractions between $C_{60}O$. This 50% heuristic derives from $\varepsilon_{CC} > 1.5$ being required to see micellization, versus $\varepsilon_{AA} = 1$ plus any contributions from BB, AB, and electrostatic interactions (if present). Both models capture qualitative micellization, and the differences between the two models demonstrate that micellization transition temperatures and the structural details of the micelles can be further fine-tuned through the inclusion of detailed electrostatic interactions. To further put these results

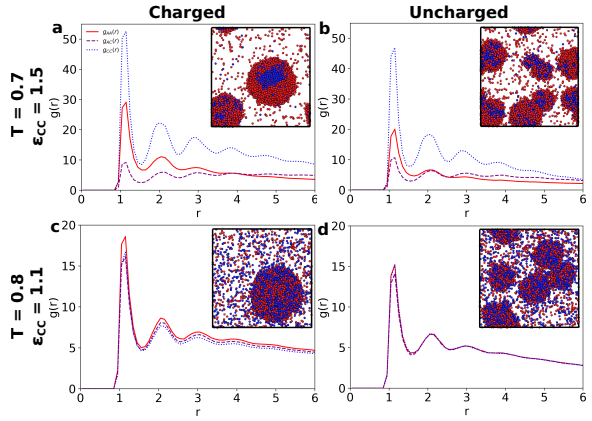


Figure 9: Comparison of Models 6 and 7 for varying temperature and ε_{CC}. a) Model 6: $T = 0.7$ and $\varepsilon_{CC} = 1.5$, b) Model 7: $T = 0.7$ and $\varepsilon_{CC} = 1.5$, c) Model 6: $T = 0.8$ and $\varepsilon_{CC} = 1.1$, d) Model 7: $T = 0.8$ and $\varepsilon_{CC} = 1.1$.

into context, we reiterate that all of the observations presented are for *implicit water*: The drag and thermal fluctuations of water are modeled through Equations 1 and 2, and hydrophobic-hydrophilic interactions are implied through the choice of ε and q. These results demonstrate both that (1) micellization of C_{60} by $C_{60}O$ can phenomenalogically be described with a simple model, and (2) the conditions under which micelles form may be more readily deduced by searching parameter space, as performed here, as opposed to performing more expensive (perhaps intractably so) fully-atomistic or first-principles calculations.

5 CONCLUSIONS

Through systematic iteration of minimal models of C_{60} and $C_{60}O$, we have identified sufficient conditions for micelles of $C_{60}O$-sheathed C_{60} to form: When the net attraction between C_{60} molecules is at least 50% larger than the attraction between $C_{60}O$ molecules ($\varepsilon_{CC} > 1.5$), and when solvent thermal fluctuations are low ($T \leq 0.7$) relative to $C_{60}O$-attractions. We also find that including long-range electrostatics and a dipole moment informed by DFT calculations, the fraction of free oxides observed in solution is lower due to their assembly as the surfactant on the micelles. In sum, the minimal models of water presented here (Models 6 and 7) demonstrate fullerene micellization. These models, which equilibrate in minutes on a modern CPU or GPU, can now be used to efficiently probe questions about how micellization depends on interaction parameters beyond the present work.

Connecting the details of these models back to experimental conditions is in theory possible, but not straightforward. Small amounts of dissolved salts, for example, have a strong impact on charge screening and therefore complicate translation between models and experimental conditions. Further, the polarizability of water and fullerene molecules will determine the actual kinetics and thermodynamics of micelle formation. Nevertheless, the models developed here should provide a useful foundation for future studies of fullerene micellization in water. As one example, future studies investigating the role $C_{60}O$ and C_{60} concentrations have on micelle formation and micelle size are a natural extension of the present work. Microscopy or zeta potential measuremnts that deduce cluster sizes and surfactant ($C_{60}O$) thickness would also provide natural extensions and data for validation of this work. Further calibration of the interaction parameters in Models 6 and 7 can be performed with experimental measurements of colloidal size distributions, and these calibrated models would provide additional capabilities for predicting temperature ranges over which micellization is optimized or thermodynamically unfavorable. We expect the use of simplified models such as the ones presented here to aid in the interpretation of experiments of aqueous fullerene dispersion and ultimately inform more efficient methods for incorporating fullerenes into the advanced materials of the future.

6 REFLECTIONS

Participating in the Blue Waters Student Internship Program in 2016 kick-started my (KN) computational science career and passion for data science. It all started during the two-week petascale institute at the University of Illinois at Urbana-Champaign where I, alongside twenty-five other aspiring computational scientists, learned programming basics and how to submit jobs to a supercomputer, including those comprising the beginnings of this project. We discussed our own projects and brainstormed ideas for improving their effectiveness by employing complex HPC concepts. Through this experience, I also made invaluable networking connections with scientists from both the Shodor Education Foundation and the National Center for Supercomputing Applications. The computing community is close-knit and incredibly supportive, for which I am very grateful to be a part. Working on this project and participating in the Blue Waters program gave me a glimpse at the abundant opportunities available in high-performance computing. Scientists from many different backgrounds (biology, physics, engineering, climatology) attended the institute to develop their skills and advance their respective fields.

The largest challenge in this project was completing and submitting this manuscript! While the basic science was accomplished during the 2016 Blue Waters internship, balancing data analysis, learning scientific writing, and doing scientific writing with classes, varsity track and field, and engineering society involvement caused the submission to be rather delayed. Nevertheless, I am proud of the story we were able to tie together with this work, and the trajectory on which the Blue Waters internship has put me.

My ever-evolving expertise and curiosity in computation have provided me with exposure to projects in materials science, astrochemistry, and even neuroscience. The technical skills, confidence, networking connections, and data-driven mindset I developed during this experience will facilitate my future endeavours; this includes my pursuit of a Ph.D. in Neural Computation at Carnegie Mellon University, which I will begin in the Fall of 2020.

7 SOFTWARE

Job submission scripts and analysis are available in the online supplementary information.

8 ACKNOWLEDGMENTS

This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575[50]. KN and EJ thank the Shodor Education Foundation and the Blue Waters Student Internship Program for support of this work. The quantum chemical computations were conducted on the ETH High Performance Computing clusters (EULER). This material is based upon work supported by the National Science Foundation under Grant Nos. (1229709) and (1653954).

REFERENCES

[1] J. D. Fortner, D. Y. Lyon, C. M. Sayes, A. M. Boyd, J. C. Falkner, E. M. Hotze, L. B. Alemany, Y. J. Tao, W. Guo, K. D. Ausman, V. L. Colvin, and J. B. Hughes. C 60 in Water: Nanocrystal Formation and Microbial Response. *Environmental Science & Technology*, 39(11):4307–4316, jun 2005.

[2] Grigoriy V. Andrievsky, Marina V. Kosevich, Oleh M. Vovk, Vadim S. Shelkovsky, and Lyudmila A. Vashchenko. On the production of an aqueous colloidal solution of fullerenes. *Journal of the Chemical Society, Chemical Communications*, 8(12):1281, 1995.

[3] Shigeru Deguchi, Rossitza G. Alargova, and Kaoru Tsujii. Stable Dispersions of Fullerenes, C60 and C70, in Water: Preparation and Characterization. *Langmuir*, 17(19):6013–6017, sep 2001.

[4] Nikolay O. Mchedlov-Petrossyan, Vladimir K. Klochkov, and Grigoriy V. Andrievsky. Colloidal dispersions of fullerene C60 in water: some properties and regularities of coagulation by electrolytes. *Journal of the Chemical Society, Faraday Transactions*, 93(24):4343–4346, 1997.

[5] Walter A. Scrivens, James M. Tour, Kim E. Creek, and Lucia Pirisi. Synthesis of 14C-Labeled C60, Its Suspension in Water, and Its Uptake by Human Keratinocytes. *Journal of the American Chemical Society*, 116(10):4517–4518, may 1994.

[6] Jonathan A. Brant, Jérôme Labille, Jean-Yves Bottero, and Mark R. Wiesner. Characterizing the Impact of Preparation Method on Fullerene Cluster Structure and Chemistry. *Langmuir*, 22(8):3878–3885, apr 2006.

[7] J. Labille, J. Brant, F. Villiéras, M. Pelletier, A. Thill, A. Masion, M. Wiesner, J. Rose, and J.âĂŘY. Bottero. Affinity of C60 Fullerenes with Water. *Fullerenes, Nanotubes and Carbon Nanostructures*, 14(2-3):307–314, dec 2006.

[8] Alok Dhawan, Julian S. Taurozzi, Alok K. Pandey, Wenqian Shan, Sarah M. Miller, Syed A. Hashsham, and Volodymyr V. Tarabara. Stable Colloidal Dispersions of C60 Fullerenes in Water: Evidence for Genotoxicity. *Environmental Science & Technology*, 40(23):7394–7401, dec 2006.

[9] Najla Gharbi, Monique Pressac, Michelle Hadchouel, Henri Szwarc, Stephen R. Wilson, and Fathi Moussa. [60]Fullerene is a Powerful Antioxidant in Vivo with No Acute or Subacute Toxicity. *Nano Letters*, 5(12):2578–2585, dec 2005.

[10] Nicole Levi, Roy R Hantgan, Mark O Lively, David L Carroll, and Gaddamanugu L Prasad. C60-fullerenes: detection of intracellular photoluminescence and lack of cytotoxic effects. *Journal of Nanobiotechnology*, 4(1):14, 2006.

[11] Delina Y. Lyon and Pedro J.J. Alvarez. Fullerene Water Suspension (nC 60) Exerts Antibacterial Effects via ROS-Independent Protein Oxidation. *Environmental Science & Technology*, 42(21):8127–8132, nov 2008.

[12] Eva Oberdörster. Manufactured Nanomaterials (Fullerenes, C60) Induce Oxidative Stress in the Brain of Juvenile Largemouth Bass. *Environmental Health Perspectives*, 112(10):1058–1062, apr 2004.

[13] Christie M. Sayes, John D. Fortner, Wenh Guo, Delina Lyon, Adina M. Boyd, Kevin D. Ausman, Yizhi J. Tao, Balaji Sitharaman, Lon J. Wilson, Joseph B. Hughes, Jennifer L. West, and Vicki L. Colvin. The Differential Cytotoxicity of Water-Soluble Fullerenes. *Nano Letters*, 4(10):1881–1887, oct 2004.

[14] Christie M. Sayes, Andre M. Gobin, Kevin D. Ausman, Joe Mendez, Jennifer L. West, and Vicki L. Colvin. Nano-C60 cytotoxicity is due to lipid peroxidation. *Biomaterials*, 26(36):7587–7595, dec 2005.

[15] Naohide Shinohara, Takeru Matsumoto, Masashi Gamo, Arisa Miyauchi, Shigehisa Endo, Yoshitaka Yonezawa, and Junko Nakanishi. Is Lipid Peroxidation Induced by the Aqueous Suspension of Fullerene C 60 Nanoparticles in the Brains of Cyprinus carpio ? *Environmental Science & Technology*, 43(3):948–953, feb 2009.

[16] Shiqian Zhu, Eva Oberdörster, and Mary L. Haasch. Toxicity of an engineered nanoparticle (fullerene, C60) in two aquatic species, Daphnia and fathead minnow. *Marine Environmental Research*, 62:S5–S9, jan 2006.

[17] Xiaoshan Zhu, Lin Zhu, Yupeng Lang, and Yongsheng Chen. Oxidative Stress And Growth Inhibition In The Freshwater Fish Carassius Auratus Induced By Chronic Exposure To Sublethal Fullerene Aggregates. *Environmental Toxicology and Chemistry*, 27(9):1979, 2008.

[18] Randall D. Maples, Martha E. Hilburn, Befrika S. Murdianti, Rangika S. Hikkaduwa Koralege, Jason S. Williams, Satish I. Kuriyavar, and Kevin D. Ausman. Optimized solvent-exchange synthesis method for C60 colloidal dispersions. *Journal of Colloid and Interface Science*, 370(1):27–31, mar 2012.

[19] Jonathan Brant, Hélène Lecoanet, Matt Hotze, and Mark Wiesner. Comparison of Electrokinetic Properties of Colloidal Fullerenes (n -C 60) Formed Using Two Procedures âĂă. *Environmental Science & Technology*, 39(17):6343–6351, sep 2005.

[20] Martha E. Hilburn, Befrika S. Murdianti, Randall D. Maples, Jason S. Williams, Joshua T. Damron, Satish I. Kuriyavar, and Kevin D. Ausman. Synthesizing aqueous fullerene colloidal suspensions by new solvent-exchange methods. *Colloids and Surfaces A: Physicochemical and Engineering Aspects*, 401:48–53, may 2012.

[21] Befrika S. Murdianti, Joshua T. Damron, Martha E. Hilburn, Randall D. Maples, Rangika S. Hikkaduwa Koralege, Satish I. Kuriyavar, and Kevin D. Ausman. C60 oxide as a key component of aqueous C60 colloidal suspensions. *Environmental Science and Technology*, 46(14):7446–7453, 2012.

[22] Xuekun Cheng, Amy T. Kan, and Mason B. Tomson. Naphthalene Adsorption and Desorption from Aqueous C 60 Fullerene. *Journal of Chemical & Engineering Data*, 49(3):675–683, may 2004.

[23] Befrika S. Murdianti, Joshua T. Damron, Martha E. Hilburn, Randall D. Maples, Rangika S. Hikkaduwa Koralege, Satish I. Kuriyavar, and Kevin D. Ausman. C 60 Oxide as a Key Component of Aqueous C 60 Colloidal Suspensions. *Environmental Science & Technology*, 46(14):7446–7453, jul 2012.

[24] Hiroshi Noguchi and Masako Takasu. Self-assembly of amphiphiles into vesicles: A Brownian dynamics simulation. *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics*, 64(4):7, 2001.

[25] Siewert J. Marrink, Jelger Risselada, and Alan E. Mark. Simulation of gel phase formation and melting in lipid bilayers using a coarse grained model. *Chemistry and Physics of Lipids*, 135(2):223–244, 2005.

[26] Susumu Fujiwara, Takashi Itoh, Masato Hashimoto, and Ritoku Horiuchi. Molecular dynamics simulation of amphiphilic molecules in solution: Micelle formation and dynamic coexistence. *Journal of Chemical Physics*, 130(14), 2009.

[27] Guido Avvisati, Teun Vissers, and Marjolein Dijkstra. Self-Assembly of Patchy Colloidal Dumbbells. pages 1–10, dec 2014.

[28] Jacob N Israelachvili, D John Mitchell, and Barry W Ninham. Theory of self-assembly of hydrocarbon amphiphiles into micelles and bilayers. *Journal of the Chemical Society, Faraday Transactions 2*, 72:1525, 1976.

[29] R Kubo. The fluctuation-dissipation theorem. *Reports on Progress in Physics*, 29(1):306, jan 1966.

[30] Robert C. Rizzo and William L. Jorgensen. OPLS all-atom model for amines: Resolution of the amine hydration problem. *Journal of the American Chemical Society*, 121(11):4827–4836, 1999.

[31] H A Lorentz. Ueber die Anwendung des Satzes vom Virial in der kinetischen Theorie der Gase. *Annalen der Physik*, 248(1):127–136, 1881.

[32] Daniel Berthelot. Sur le mélange des gaz. *Compt. Rendus*, 126:1703–1706, 1898.

[33] Ch. Girard, Ph. Lambin, A. Dereux, and A. A. Lucas. van der Waals attraction between two C60 fullerene molecules and physical adsorption of C60 on graphite and other substrates. *Physical Review B*, 49(16):11425–11432, apr 1994.

[34] Hojin Kim, Dmitry Bedrov, Grant D. Smith, and Salt Lake City. Molecular Dynamics Simulation Study of the Influence of Cluster Geometry on Formation of C 60 Fullerene Clusters in Aqueous Solution. *Journal of Chemical Theory and Computation*, 4(2):335–340, 2008.

[35] Trung Dac Nguyen, Carolyn L Phillips, Joshua a. Anderson, and Sharon C Glotzer. Rigid Body Constraints Realized in Massively-parallel Molecular Dynamics on Graphics Processing Units. *Computer Physics Communications*, 182(11):2307–2313, nov 2011.

[36] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, m. Cococcioni, I. Dabo, A. Dal Corso, S. Fabris, G. Fratesi, S. de Gironcoli, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A. P. Seitsonen, A. Smogunov, P. Umari, and R. M. Wentzcovitch. Quantum ESPRESSO: a modular and open-source software project for quantum simulations of materials. 2009.

[37] John P. Perdew, Kieron Burke, and Matthias Ernzerhof. Generalized Gradient Approximation Made Simple. *Physical Review Letters*, 77(18):3865–3868, 1996.

[38] G. Kresse and D. Joubert. From ultrasoft pseudopotentials to the projector augmented-wave method. *Physical Review B*, 59(3):1758–1775, 1999.

[39] Richard F. W. Bader. Atoms in Molecules. In *Encyclopedia of Computational Chemistry*. John Wiley & Sons, Ltd, Chichester, UK, apr 2002.

[40] Graeme Henkelman, Andri Arnaldsson, and Hannes Jónsson. A fast and robust algorithm for Bader decomposition of charge density. *Computational Materials*

Science, 36(3):354–360, 2006.

[41] Edward Sanville, Steven D. Kenny, Roger Smith, and Graeme Henkelman. Improved grid-based algorithm for Bader charge allocation. *Journal of Computational Chemistry*, 28(5):899–908, apr 2007.

[42] Joshua A. Anderson, Jens Glaser, and Sharon C. Glotzer. HOOMD-blue: A Python package for high-performance molecular dynamics and hard particle Monte Carlo simulations. *Computational Materials Science*, 173(October 2019):109363, 2020.

[43] William G. Hoover. Canonical Dynamics: Equilibrium Phase-space Distributions. *Physical Review A*, 31(3):1695–1697, 1985.

[44] A. Denner, S. Dittmaier, M. Roth, and L. H. Wieders. Complete electroweak O(alpha) corrections to charged-current e+e- –> 4fermion processes. *The Journal of Chemical Physics*, 101(5):4177–4189, feb 2005.

[45] J. Cao and G. J. Martyna. Adiabatic path integral molecular dynamics methods. II. Algorithms. *The Journal of Chemical Physics*, 104(5):2028–2035, feb 1996.

[46] David N. Lebard, Benjamin G. Levine, Philipp Mertmann, Stephen A. Barr, Arben Jusufi, Samantha Sanders, Michael L. Klein, Athanassios Z. Panagiotopoulos, A Barr, Arben Jusufi, Samantha Sanders, and L Klein. Self-assembly of coarse-grained ionic surfactants accelerated by graphics processing units. *Soft Matter*, 8(8):2385–2397, 2012.

[47] W. C. Swope, H. C. Andersen, P. H. Berens, and K. R. Wilson. A Computer Simulation Method for the Calculation of Equilibrium Constants for the Formation of Physical Clusters of Molecules: Application to Small Water Clusters. *Journal of Chemical Physics*, 76(1):637–649, 1982.

[48] William Humphrey, Andrew Dalke, and Klaus Schulten. VMD: Visual molecular dynamics. *Journal of Molecular Graphics*, 14(1):33–38, feb 1996.

[49] J D Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[50] John Towns, Timothy Cockerill, Maytal Dahan, Ian Foster, Kelly Gaither, Andrew Grimshaw, Victor Hazlewood, Scott Lathrop, Dave Lifka, Gregory D. Peterson, Ralph Roskies, J. Ray Scott, and Nancy Wilkens-Diehr. XSEDE: Accelerating Scientific Discovery. *Computing in Science & Engineering*, 16(5):62–74, sep 2014.

Performance Analysis of the Parallel CFD Code for Turbulent Mixing Simulations

Tulin Kaman*
Department of Mathematical Sciences
University of Arkansas
Fayetteville, AR
tkaman@uark.edu

Alaina Edwards
Department of Mathematical Sciences
University of Arkansas
Fayetteville, AR
aje004@uark.edu

John McGarigal
Department of Mechanical Engr.
University of Arkansas
Fayetteville, AR
jamcgari@uark.edu

ABSTRACT

Understanding turbulence and mixing due to the hydrodynamic instabilities plays an important role in a wide range of science and engineering applications. Numerical simulations of three dimensional turbulent mixing help us to predict the dynamics of two fluids of different densities, one over the other. The focus of this work is to optimize and improve the computational performance of the numerical simulations for the compressible turbulent mixing on Blue Waters, the petascale supercomputer at the National Center for Supercomputing Applications. In this paper, we study the effect of the programming models on time to solution. The hybrid programming model, which is a combination of parallel programming models, becomes a dominant approach. The most preferable hybrid model is the one that involves the Message Passing Interface (MPI), such as MPI + Pthreads, MPI + OpenMP, MPI + MPI-3 shared memory programming, and others with accelerator support. Among all choices, we choose the hybrid programming model that is based on MPI + OpenMP. We extend the purely MPI parallelized code with OpenMP parallelism and develop the hybrid version of the code. This new hybrid implementation of the code is set up in a way that multiple MPI processes handle the interface propagation, whereas multiple OpenMP threads handle the high order weighted essentially non-oscillatory numerical scheme.

KEYWORDS

Turbulent Mixing, Numerical Simulations, Performance Analysis, Distributed/Shared memory programming

1 INTRODUCTION

Turbulent mixing flows arise in a wide range of science and engineering applications, from climate studies to all forms of fusion, whether the confinement is inertial, gravitational or magnetic. The numerical simulations help us to understand the dynamics of turbulent mixing. Turbulent mixing due to Rayleigh-Taylor (RT) instability arises at the interface between two fluids of different densities whenever the pressure gradient opposes the density gradient.

These problems are deeply multiscale. The level of scales that are desired to be resolved identifies the characteristic properties of the numerical approach, such as Direct Numerical Simulations (DNS), Large Eddy Simulations (LES), and Reynolds Averaged Navier Stokes (RANS). With the power of today's HPC systems, resolving all turbulence length scales is handled by DNS [14]. RANS resolves length scales sufficient to specify the problem geometry. Among these three approaches, DNS has the highest computational cost, and RANS has the lowest. The use of LES reduces the computational cost of the DNS and resolves some but not all length scales. LES was first proposed by Smagorinsky [21] for the study of the dynamics of the atmosphere's general circulation. In LES, the unresolved smaller-scale motions are modeled by subgrid scale (SGS). The multi-species compressible Navier-Stokes equation, filtered at a grid level, is solved, so that the LES defines SGS terms (such as the Reynolds stress) as a source. These source terms are modeled as gradient diffusion terms, and otherwise coefficients such as turbulent viscosity, mass, and thermal diffusivity are recovered in a dynamic manner from the solution itself. This is called a dynamic SGS model. The missing coefficients are computed locally in the simulation [2].

Front tracking (FT) is the technique of storing and dynamically evolving a meshed front that partitions a simulation domain into two or more regions, each representing a different material, or physics model. Front tracking is the unique method presently demonstrated to avoid systematic errors in an important class of problems revolving around turbulent mixing [5]. The sharp resolution of interfaces and steep gradients occurs in a variety of applications, such as primary breakup of a liquid jet [1], forecasting of cloud boundaries [7], target design of muon collider in high energy particle accelerators [4], and electrocardia [23]. The FT/LES/SGS combination has previously been validated for macro, meso, and micro scale observables. By this, This means that diagnostics with comparison to experimental data have been applied to assess the solution accuracy of the turbulent mixing flow at the macro/meso/micro length scales. Thus, the simulations have achieved agreement with experimental measurements in the overall size of the mixing zone (macro), the coherent bubble structure within it (meso), and molecular level mixing (micro) as recorded by chemical reactions [5, 12].

The simulation package, FronTier, has the implementation of all these algorithms. FronTier supports a range of physics, including compressible and incompressible flow, turbulence models, fluid-structure interactions, phase transitions, and crystal growth, each with its own validation and verification studies [3]. It is parallelized with a tensor product domain decomposition. MPI is used to pass states and interface data from one processor to another. FronTier

*Corresponding Author

https://doi.org/10.22369/issn.2153-4136/12/1/7

has adopted object oriented programming. Many major front tracking functionalities have been modularized to allow users to call them with a minimal knowledge of internal operation and coding. An API to modularize the front tracking and to make it available to other simulation codes is constructed [13]. We extend the purely MPI parallelized code with OpenMP parallelism and develop the hybrid version of the code. The focus is to optimize and improve the computational performance and increase the scalability to perform high resolution numerical simulations efficiently on high performance computing systems.

The organization of this paper is as follows. In Section 2, we describe the model problem, Rayleigh-Taylor instability. In Section 3, we present the strong and weak scaling analysis of the purely MPI parallel version of the code and introduce the profiling tool Tuning and Analysis Tool (TAU) that is used to analyze the runtime behavior of the program. We show how to instrument the FronTier code using TAU and present performance results. In Section 4, we introduce the hybrid programming model, which is a combination MPI + OpenMP parallel programming model and compare the purely MPI and hybrid models. Section 5 presents reflections on how the research activities as Blue Waters Interns influence the students' future careers.

2 PROBLEM DESCRIPTION

The study of the turbulent mixing problem in Rayleigh-Taylor aims for macro level validation and to predict the overall dimensionless growth rate of the mixing zone. The growth rate is defined by the formula $h_b = \alpha_b A g t^2$ for the penetration distance h_b of the light fluid into the heavy fluid, g being the acceleration force that defines the instability, and $A = (\rho_1 - \rho_2)/(\rho_1 + \rho_2)$ the dimensionless buoyancy correction to gravity, depending on the density difference between the two fluids. Here, ρ_1 denotes the heavy fluid density, and ρ_2 denotes the light fluid density. The determination of the growth rate α_b has been the source of considerable interest. The numerical simulations are conducted to predict the quantity of interest on the growth rate of the mixing zone. The uncertainty quantification analysis associated with initial conditions, the sensitivity analysis to the parameters such as the initial mass diffusion layer thickness, and the effect of initial conditions and parameters to the quantity of interest α_b were studied in comparison with experimental data and presented in [9–12, 24]. The problem with single-mode, multi-mode, and random initial perturbations is presented in Figure 1. Here, the validation and verification studies are out of the scope of this paper. The focus is to optimize and improve the computational performance of the FronTier software package for the Rayleigh-Taylor turbulent mixing problem on the Blue Waters petascale supercomputer.

For the numerical simulations of multiphase flows, one of the advantages of the front tracking method is in dealing with the contact discontinuities. The front tracking method is used to solve the conservation laws with discontinuities between fluids. The mathematical formulation is based on the filtered Navier-Stokes equations for the multiphase flows [10]. These equations are the governing equations of LES simulations. In the equations of continuity, momentum, energy, and concentration (1) – (4), the variables that have been filtered on the grid scale are denoted by the overbar. There

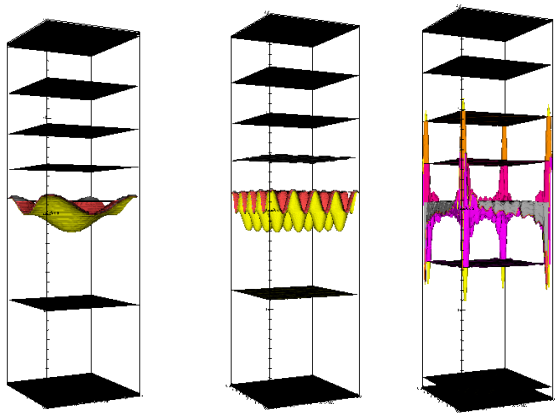


Figure 1: Images of the single-mode, uniform and random multi-mode Rayleigh-Taylor instabilities.

is also a density-weighted filtering operation, which is denoted by the tilde. The Favre-filtered continuity equation (1) is obtained by first applying the grid scale onto the continuity equation

$$\frac{\partial \overline{\rho}}{\partial t} + \frac{\partial \overline{\rho v_i}}{\partial x_i} = 0,$$

then the density-weighted filtering $\widetilde{v_i} = \dfrac{\overline{\rho v_i}}{\overline{\rho}}$.

For the compressible flows, the Favre-filtered continuity, momentum, energy, and concentration equations are obtained as

$$\frac{\partial \overline{\rho}}{\partial t} + \frac{\partial \overline{\rho}\widetilde{v_i}}{\partial x_i} = 0 \tag{1}$$

$$\frac{\partial \overline{\rho}\widetilde{v_j}}{\partial t} + \frac{\partial (\overline{\rho}\widetilde{v_i}\widetilde{v_j} + \overline{p}\delta_{ij})}{\partial x_i} = \frac{\partial \overline{d_{ij}}}{\partial x_i} \tag{2}$$

$$\begin{aligned}\frac{\partial \overline{E}}{\partial t} + \frac{\partial (\overline{E} + \overline{p})\widetilde{v_i}}{\partial x_i} &= \frac{\partial \overline{d_{ij}}\widetilde{v_j}}{\partial x_i} + \frac{\partial}{\partial x_i}\left(\overline{\kappa}\frac{\partial \widetilde{T}}{\partial x_i}\right) \\ &\quad + \frac{\partial}{\partial x_i}\left((\widetilde{H_h} - \widetilde{H_l})\overline{\rho}\widetilde{D}\frac{\partial \widetilde{\Psi}}{\partial x_i}\right),\end{aligned} \tag{3}$$

$$\frac{\partial \overline{\rho}\widetilde{\Psi}}{\partial t} + \frac{\partial \overline{\rho}\widetilde{\Psi}\widetilde{v_i}}{\partial x_i} = \frac{\partial}{\partial x_i}\left(\overline{\rho}\widetilde{D}\frac{\partial \widetilde{\Psi}}{\partial x_i}\right). \tag{4}$$

where $\overline{\rho}, \widetilde{v_i}, \overline{E}, \overline{p}$, and $\widetilde{\Psi}$ are the filtered variables for total mass density, the velocity, the total specific energy, the pressure, and the mass fraction. The total specific energy is

$$\overline{E} = \overline{\rho e} + \overline{\rho v_k}^2/2.$$

$\widetilde{H_h}$ and $\widetilde{H_l}$ are the partial specific enthalpy of each species defined by

$$\widetilde{H_h} = \widetilde{e_h} + \frac{\overline{p}}{\overline{\rho}}, \qquad \widetilde{H_l} = \widetilde{e_l} + \frac{\overline{p}}{\overline{\rho}},$$

where $\widetilde{e_h}$ and $\widetilde{e_l}$ are the specific internal energy of each species. $\widetilde{T}, \overline{\kappa}$, and $\widetilde{D}$ are the filtered temperature, the heat conductivity, and the kinematic mass diffusivity. The viscous stress tensor, d_{ij}, in momentum and energy equations is expressed as

$$\overline{d_{ij}} = \overline{\nu_d}\left(\left(\frac{\partial \widetilde{v_i}}{\partial x_j} + \frac{\partial \widetilde{v_j}}{\partial x_i}\right) - \frac{2}{3}\frac{\partial \widetilde{v_k}}{\partial x_k}\delta_{ij}\right),$$

where $\overline{\nu_d} = \overline{\rho\nu_k}$ is the filtered dynamic viscosity.

The stable and higher order WENO (Weighted Essentially Non-Oscillatory) scheme [19] is used for solving the Favre-averaged Navier-Stokes equations. In this section, the WENO scheme is briefly explained. The main features of the WENO finite difference methods, finite volume methods, and the discontinuous Galerkin finite element methods for computational fluid dynamics can be found in Shu's paper [20]. The flux-averaged WENO method uses local Lax-Friedrichs flux-splitting and a characteristic decomposition of the variables and fluxes. The fluxes in x, y, and z are calculated separately. High order accurate and non-oscillatory scheme flux reconstruction uses a convex combination of k candidate stencils [8]. For $k = 3$, the fifth $(2k - 1)$ order finite difference WENO scheme approximates the derivative $\mathbf{F}(\mathbf{U})_x$ at a point x_i,

$$\mathbf{F}(\mathbf{U})_x|_{x=x_i} \approx \frac{1}{\Delta x}(\hat{F}_{i+1/2} - \hat{F}_{i-1/2}) \tag{5}$$

where $\mathbf{U}$ is the state vector, $\mathbf{F}(\mathbf{U})$ is the flux, and $\hat{F}_{i+1/2}$ and $\hat{F}_{i-1/2}$ are the right and left fluxes. The fifth order WENO scheme uses three stencils,

$$\hat{F}_{i+1/2} = \sum_{j=1}^{3} \omega_i \hat{F}^{(j)}_{i+1/2}$$

with three third order fluxes $\hat{F}^{(j)}_{i+1/2}$ and the nonlinear weights ω_i.

For hyperbolic conservation equations, the nonlinear part of WENO is carried out in local characteristics fields. The implementation starts by computing the average state $\bar{U}_{i+1/2,j,k}$ using the average, then the left and right eigenvectors and eigenvalues of the Jacobian $F'(\bar{U}_{i+1/2,j,k})$ at the average state. One can project the conservative fields and the fluxes onto the local characteristic fields using the left eigenvectors matrix and compute the left and right fluxes in characteristic field. Then, project back the numerical fluxes in the physical space using the right eigenvector matrix. We perform the same steps for the other two directions in y and z using the average states $\bar{U}_{i,j+1/2,k}$ and $\bar{U}_{i,j,k+1/2}$. In the next section, we will observe that WENO flux computation is the most time consuming in our simulations.

3 PERFORMANCE STUDIES

Our primary goal is to achieve performance improvements of the numerical simulations for hydrodynamic instabilities. We start with identifying the parts of the code that are time consuming. For this, we take advantage of the available performance analysis tools on Blue Waters. The first tool used for profiling is the Cray Performance and Analysis Tools (CPMAT). CPMAT is equipped with several components used in preparing any project for performance analysis. CPMAT is able to gather data during the program execution, and the data can be processed and analyzed for presentation to the user on its own graphical user interface, Cray Apprentice2. The details of how the code was prepared and how the analysis was carried out can be found in Blue Waters' user guide, but, in essence, any performance analysis process has three main steps: code instrumentation, execution, and data analysis. Apprentice2 is used for visualizing and exploring the data for analysis.

There are other profiling tools on the system, such as PAPI, PerfSuite, and TAU [15]. Among these profiling tools, we choose TAU, which is a comprehensive code profile tool with additional features

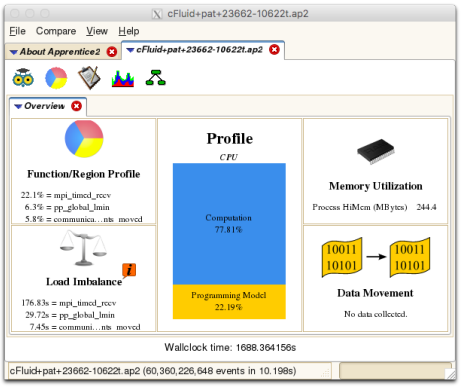


Figure 2: CPMAT Apprentice2 visualization.

for our performance analysis. In Section 3.3, the portable, robust, and parallel scalable TAU tool [18] is introduced for the performance instrumentation, measurement, analysis, and visualization.

The Blue Waters system is a Cray XE/XK hybrid machine composed of AMD 6276 "Interlagos" processors and NVIDIA GK110 (K20X) "Kepler" accelerators, all connected by the Cray Gemini torus interconnect [15]. The XE node has 2 Interlagos processors, and each processor has 16 bulldozer cores, as shown in Figure 3. Each bulldozer core's memory is 4GB, and the total node memory is 128GB. In the distributed memory parallel programming model with MPI, we observe that the memory footprint per integer core is enough to fit in memory, and we could use all 32 integer cores available on an XE node. We vary the number of MPI processes per node by setting the "-N" parameter in a job script file to investigate the effect of the processors per node (ppn) on the runtime. In Table 1, the time to solution on problem size $64 \times 64 \times 256$ with different processors per node is presented. The system default task placement for MPI processes is used in pure MPI runs. The efficiency per MPI process is virtually unaffected when changing the processors per node from 32 to 8 integer cores, and we observe a 6% loss of efficiency using 4 processors on 8 nodes. See Table 1.

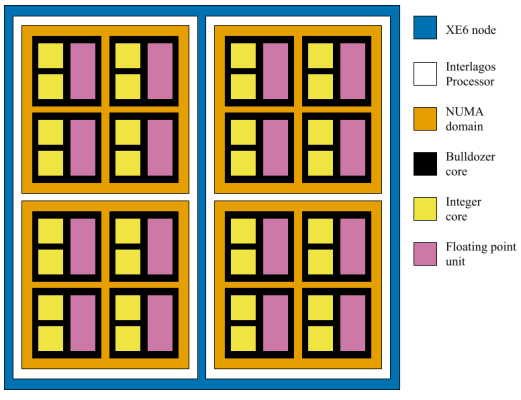


Figure 3: Cray XE6 node type on Blue Waters. Courtesy of Aaron Weeden, Blue Waters Petascale Institute notes.

Table 1: Time to solution on 32 MPI processes. #PPN is the processes per node.

#Node	#PPN	Time (seconds)
1	32	254
2	16	254
4	8	252
8	4	237

The runtime behavior is investigated by running weak and strong scaling analyses. The simulations are performed on a domain $1 \times 1 \times 4$ cm with a single grid spacing $\Delta x = \Delta y = \Delta z$. The number of grids in the z direction is four times the number of the grids in the x and y directions. The weak and strong scaling analyses are all performed using 32 processors per node in pure MPI jobs.

3.1 Weak Scaling

We conduct a weak scaling study and run simulations on four different grids: $64 \times 64 \times 256$, $128 \times 128 \times 512$, $256 \times 256 \times 1{,}024$, and $512 \times 512 \times 2{,}048$, using 32, 256, 2,048, and 16,384 cores, respectively, so that the amount of computation remains constant per core. The problem size on each MPI process is fixed and has 32 grid points in each direction. The runtimes for these problems are reported in Table 2, and they include both computation and communication. The explicit nature of the algorithm described in Section 2 contributes to the very good weak scaling as shown in Figure 4. The results for weak scaling indicate that the amount of communication increases 5% from 32 cores to 16,384 cores due to the communication overhead.

Table 2: Weak scaling for RT simulations. The grid resolution per MPI process is $32 \times 32 \times 32$.

Grid	#Processes	Actual Time to Solution	Ideal Time to Solution
$64 \times 64 \times 256$	32	254	254
$128 \times 128 \times 512$	256	263	254
$256 \times 256 \times 1{,}024$	2,048	266	254
$512 \times 512 \times 2{,}048$	16,384	269	254

3.2 Strong Scaling

To do a strong scaling analysis, we fix the total problem size while the resources are increased. The resolution of the computational grids $64 \times 64 \times 256$ (coarse) and $256 \times 256 \times 1{,}024$ (medium) run with a number of processes from 32 to 256 and from 1,024 to 8,192, respectively. Table 3 shows that the efficiency results drop to below 65% and 50% for the coarse and medium meshes. The simulations of Rayleigh-Taylor instability on the coarse and medium grid resolution are performed on processes with 2 threads (hybrid).

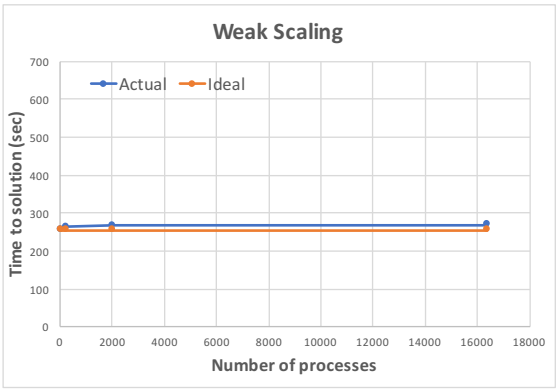


Figure 4: Runtimes under weak scaling.

Table 3: Strong scaling for RT simulations at grid resolutions of the coarse (C) and the medium (M) meshes.

#Processes		Actual Time to Solution		Ideal Time to Solution		Efficiency	
C	M	C	M	C	M	C	M
32	1,024	254	424	254	424	100%	100%
64	2,048	153	266	127	212	83%	80%
128	4,096	81	200	63.5	106	79%	53%
256	8,192	51	113	32.75	53	63%	47%

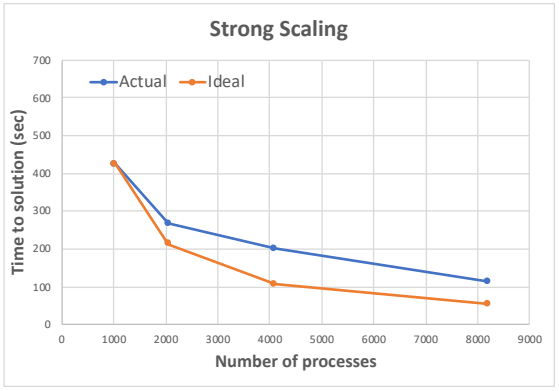


Figure 5: Runtimes under strong scaling for the medium mesh.

3.3 Profiling and Performance Analysis

In this section, we introduce the portable, robust, and parallel scalable TAU tool [18], which is used for performance instrumentation, measurement, analysis, and visualization. There are several options for instrumentation to observe the performance measurement, such as source-based, preprocessor-based, compiler-based, wrapper library-based, binary, interpreter-based, component-based, virtual machine-based, multi-level selective instrumentation, and TAU_COMPILER. The details of the instrumentation options are presented in [18].

Profiling consists of three stages: (i) compiling and linking the program with profiling enabled, (ii) executing the program to generate a profile data file, and (iii) running the profiler to analyze the data. Without any code modification, by compiling the program with a debug symbol ("-g "), a code developer can extract performance data measurements with minimal effort. TAU supports parallel profiling. Automatic instrumentation of the code using TAU's compilers (tau_cxx, tau_cc, tau_f90) and the visualization tool (ParaProf) help users to collect, analyze and visualize the performance data on thousands of processes. The TAU measurement system provides a profile data structure for each node/context/thread. Once compiling and building the executable with TAU compilers is done, we execute the new program to generate the profile data for each MPI process. The production of parallel profiles for thousands of processes requires an analysis tool to handle the performance information. TAU's scalable parallel performance profile analysis tool is called *ParaProf*. ParaProf provides a graphical interface to display all performance analysis results. ParaProf's 3D visualization option shows the spread of performance data across routines and processes. Figure 6 shows the profile data for the numerical simulation of Rayleigh-Taylor Instability. It helps in interpreting the performance data and presenting the time spent for each routine and process at once. TAU supports several types of performance profiles, such as flat, callpath, callsite, and phase profiles. The flat profile helps us to learn more about the time spent in an event, exclusive/inclusive, number of calls, and number of child calls. The profiling shows how much total time was spent in each routine. The exclusive and inclusive times show the statistics for each function. The exclusive time is the amount of time spent within that function, excluding the time spent in all child functions called from that function. The inclusive time is the amount of time spent within that function and all its child functions. The mean inclusive and exclusive times for a parallel test are presented in Figure 7. The bar graphs show the spread of performance data across routines on each process. In Figure 7, it is observed that the most time-consuming (blue bar graph, 34%) is the WENO flux computation. Each bar shows the mean exclusive time for routines and gives us an idea of how much time was spent in different routines. In the Comparison Window, Figure 8, we compare four coarse grid runs with a number of processes from 32 to 256. We have taken several steps to optimize the part of the code and work on the enhancement of the computational performance of the WENO flux computation in the `weno5_get_flux` routine.

4 HYBRID PARALLEL SCHEME

Code developers investigate the fastest programming model to handle computationally expensive simulations on clusters. An efficient programming model of the clusters of shared memory parallelism (SMP) is needed to handle large scale simulations. The parallel programming model could be purely MPI parallel or hybrid, depending on the application code. Hybrid programming with the MPI parallel scheme for internode communications and a shared memory programming for intranode communication is a preferable approach [6]. Available programming options for the shared memory parallelism are MPI-3, OpenMP, and OpenMP 4.0 / OpenACC for accelerator support. We investigate the usage of shared memory parallelism within the MPI processes for the best performance. Among the shared programming model options, we choose OpenMP for the intranode communication. OpenMP's application programming interface (API) supports the shared memory multiprocessing programming in C, C++, and Fortran. It is available since 1997 and is being actively developed to standardize directive-based, multi-language, high-level parallelism that is highly scalable and portable.

We first assess the performance of the code on the Blue Waters HPC platform using four different compilers: Cray, PGI, GNU, and Intel. The compiling and linking is performed using wrappers such as "ftn" for Fortran, "cc" for C, and "CC" for C++. To invoke the compiler, we set the programming environment corresponding to the specific compiler suite. Using the wrapper scripts and the compilers' default options provided on the system, a difference in times is observed. We observe that the PGI (Portland Group) compiler performed the best among all the compiler suites, as shown in Figure 4. There are many compiler options that can be used in the compilation process, and we use the default options that come with the wrappers. For the OpenMP directives and pragmas, "-h omp" for Cray, "-mp=nonuma" for PGI, and "-fopenmp" for GNU and Intel are provided additionally to the wrapper scripts for compiling the newly developed OpenMP implementation.

Table 4: Compiler performance on the application code.

#Processes	MPI Distribution	Time (seconds)			
		Cray	*PGI*	*GNU*	*Intel*
2,048	$8 \times 8 \times 32$	285	250	266	263
4,096	$8 \times 8 \times 64$	208	192	200	198
8,192	$16 \times 16 \times 32$	104	88	113	90

On Blue Waters, the maximum number of threads per node is 32. When running a hybrid (MPI+OpenMP) program, we first set the number of threads per process using the environment variable OMP_NUM_THREADS. In addition, the number of threads per process should be set using the depth parameter ("-d") in the run command. The depth parameter sets the number of OpenMP threads per MPI task, and it should have the exact same value as the environment variable OMP_NUM_THREADS. We specify the total number of MPI tasks for the job using the "-N" parameter, and the value for -N multiplied by the value for -d should not exceed 32 on Blue Waters.

The implementation of the WENO scheme described in Section 2 has a parallel region where we calculate the local eigenvalues, average state, right eigenvector matrix R_{mid}, and its left counterpart $L_{mid} = R^{-1}_{mid}$ at the mid-points. We transform the conservative fields, its differences, and flux differences to local characteristic field by multiplying them with L_{mid}, and we compute the numerical fluxes in each characteristic field. The last step is to project back the numerical fluxes in the physical space by multiplying them with the right eigenvector matrix R_{mid}. In this part of the code, the variables are scoped by using private and shared OpenMP data scope attribute clauses with the `parallel` directive.

In Figure 9, the time to solution using purely MPI and hybrid parallel schemes are compared. In the hybrid parallel scheme, the

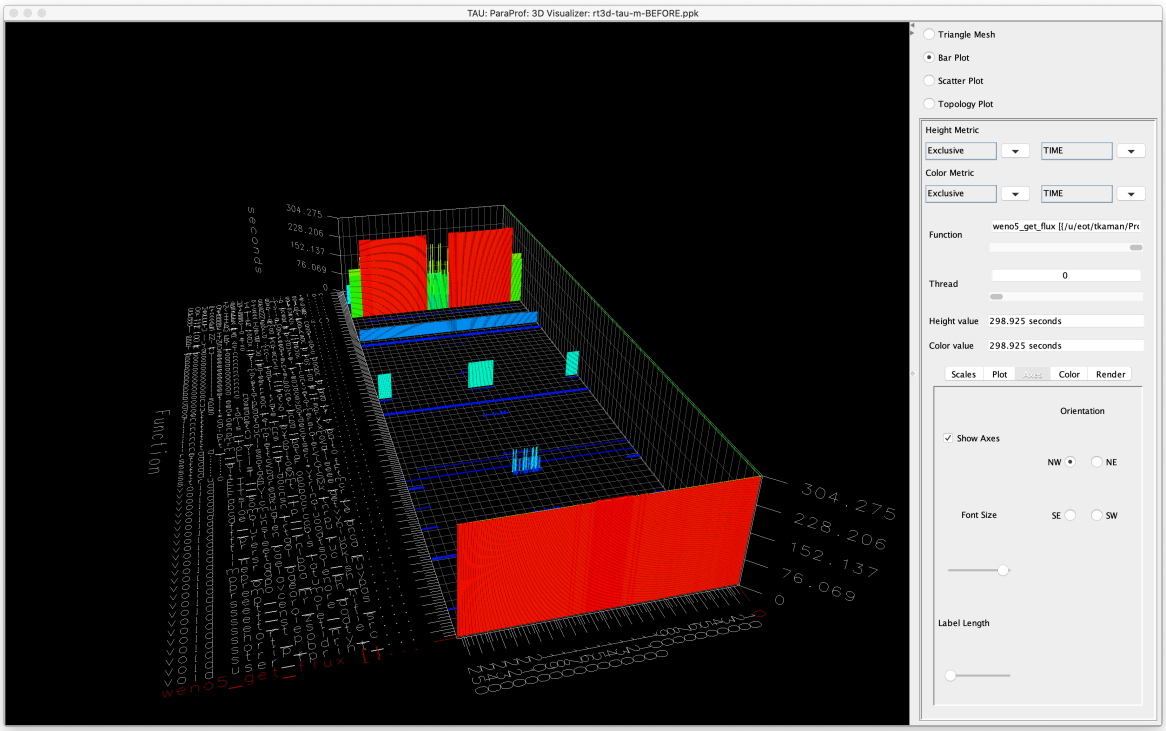


Figure 6: TAU's ParaProf 3D visualization shows the spread of performance data across routines and processes.

number of threads in OpenMP parallelism is set to two. The number of threads is controlled by setting the OMP_NUM_THREADS environment variable and giving the depth with "-d" option in running. For the hybrid case, the jobs run with 16 MPI processes per node and 2 OpenMP threads per process ("-N 16 -d 2 "). Figure 10 shows the performance improvement in the hybrid case with the inclusive time on the coarse grid. The mean exclusive time spent in the `weno5_get_flux` routine dropped from 302 seconds to 242 seconds, and the mean inclusive time is reduced by 20%. On the medium grid, we observe the pure MPI model is faster than the hybrid model. The newly developed version of the program with two OpenMP threads inside of each MPI task is slower than the pure MPI version as shown in Figure 11.

5 INTERNSHIP REFLECTION

The goal of the project is not only to investigate the effect of the parallel programming models on time to solution for the numerical simulations of compressible turbulent mixing, but also to engage the undergraduate students in petascale computing research in the area of computational fluid dynamics. The Blue Waters interns, Edwards and McGarigal, had little-to-no experience in Unix, programming in C, or parallel computing before starting this research project. A two-week intensive Petascale Institute at the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign helped them to develop the basic skills needed to start this research. Within one year, they gained experience in the usage of Blue Waters, distributed/shared parallel programming models, visualization, and performance tools. During their internship program, they were selected to attend the International Conference for High Performance Computing, Networking, Storage, and Analysis as student volunteers. There, they were able to make many significant connections to help propel them into their future careers. Edwards and McGarigal presented their first poster at the American Physics Society Conference for Undergraduate Women in Physics, which was held at the Texas A&M University at Corpus Christi and at the 2019 Annual Meeting of the Arkansas Academy of Science (AAS), which was held at Fort Collins, respectively. Their poster at AAS received the first place undergraduate poster in computer science and was also selected to be presented at the 2019 Blue Waters Symposium.

When the mentor created two Blue Waters internship positions for the two University of Arkansas students who were interested in developing skills in modeling, simulations, and high performance computing, she planned a weekly schedule for her directed reading course. This course was a one-on-one independent study to cover the topics from hydrodynamics instabilities to parallel performance systems. The directed readings were designed to help the students to see the big picture, provide an overview of the state of the project, and guide them. The papers of Zhou [25, 26] on the basic properties of the flow, turbulence, and mixing induced by hydrodynamic instabilities, Sameer [16, 18] on a TAU user's guide, and Shu [19, 20] on numerical schemes were read throughout the first semester. In the second semester, the interns' main duties were

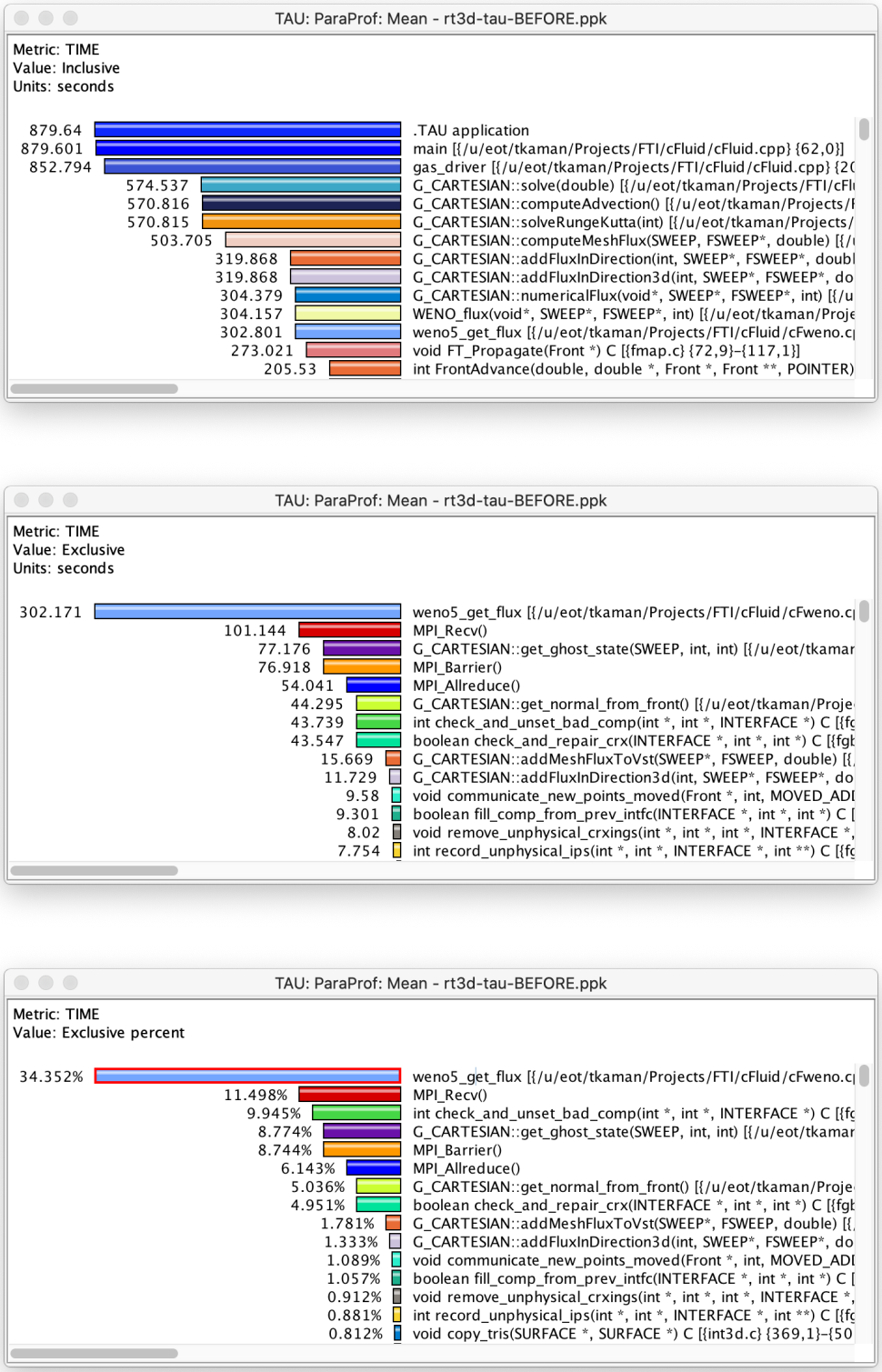


Figure 7: The mean inclusive time, exclusive time, and exclusive percent.

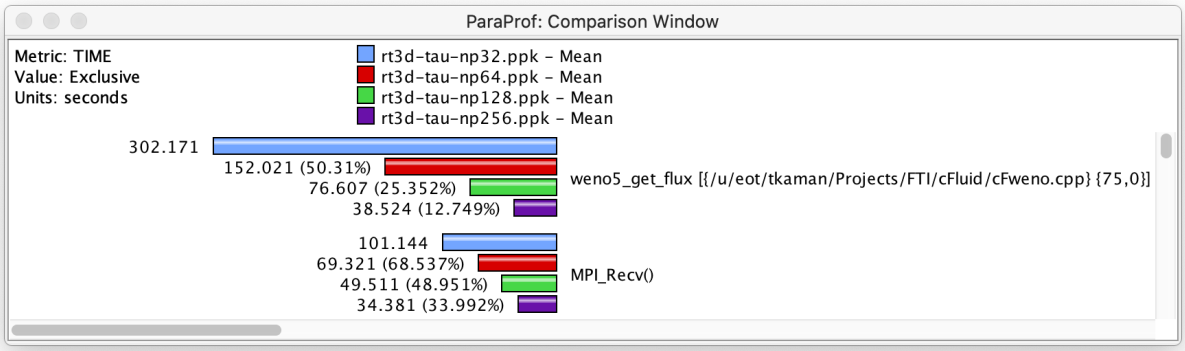


Figure 8: TAU's ParaProf Comparison Window shows the mean exclusive time.

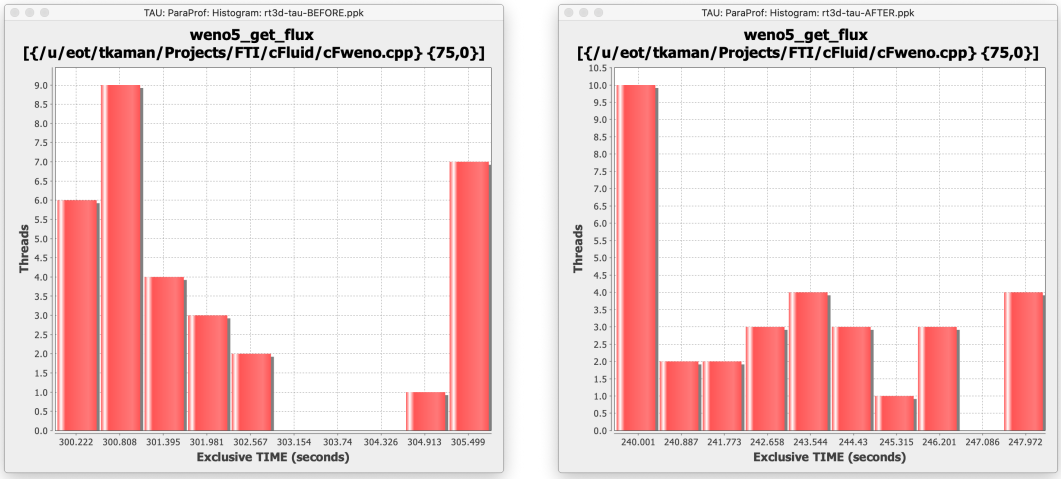


Figure 9: Comparing the performance of the `weno5_get_flux` routine using TAU's histogram function on the coarse grid.

to conduct simulations and analyze the results. To be eligible to complete the duties in a limited time, students must have met the requirements, such as being fluent in C/C++ programming, having experience with modeling and simulations, and having basic knowledge in parallel computing and computational fluid dynamics. The learning curve, the learner's performance on a task, and the number of attempts and time required for the task had taken more time than planned. In hindsight, it would have been better to add two levels of participation to the research program as learners and apprentices before their internships, as in the XSEDE EMPOWER (Expert Mentoring Producing Opportunities for Work, Education, and Research) program [17]. This way, the students would have first focused on strengthening their ability to handle challenges and taking steps on completing the assigned tasks. In a learner level, a student could have spend more time developing necessary skills to contribute to the work of Blue Waters through online tutorials, workshops, and self learning in programming. At the apprentice level, a student could have transformed the knowledge into skills and have the chance to apply the new skills with some additional trainings in debugging and performance tools to do the assigned tasks. After completing these two levels in two semesters, in their second year the students could have performed more independent work and became more fully engaged in research.

After their research experience in the computational and applied mathematics group of Kaman, the students pursue graduate studies and continue to work on computational science research projects. Edwards was one of the ten students accepted to the Oak Ridge National Laboratory Pathways to Computing Internship Program to learn and develop the next-generation explicit methods for radiation transport in astrophysics and explore programming models for GPUs supported on the fastest supercomputer in the world, Summit [22]. McGarigal started a new internship at HP as part of the test automation team, working on designing the robot framework for computers. The Blue Waters Student Internship Program helped

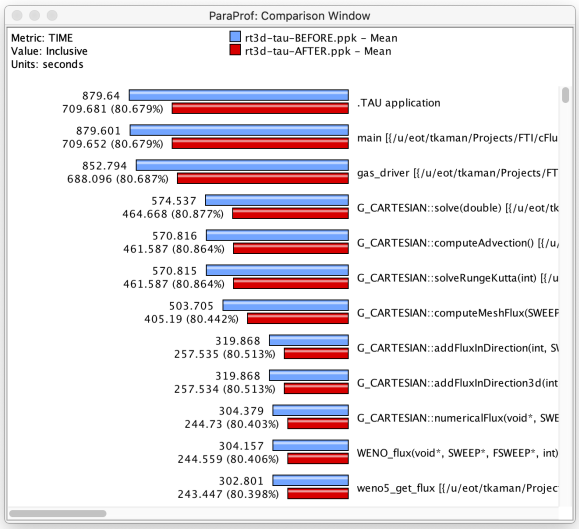


Figure 10: TAU's ParaProf Comparison Window shows the mean inclusive time before and after optimizing WENO flux on the coarse grid.

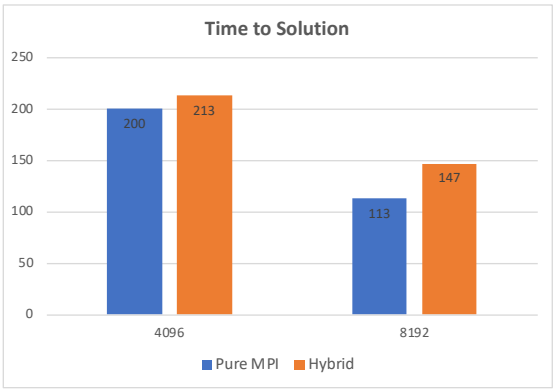


Figure 11: Comparison of purely MPI and hybrid models on the medium grid.

two University of Arkansas undergraduate students to develop strong computational skills in high performance computing and reflect their perspective of how they can advance their knowledge and skills for their future career.

6 CONCLUSIONS

Numerical simulations of turbulent mixing are computationally expensive and require efficient usage of high performance computing systems. The scalability of the purely MPI application code shows very good weak, and acceptable strong, scalability properties. We collect performance data to identify the most time consuming parts of the application code using the performance measurement and analysis tool TAU, whose performance system identifies that the high order accurate weighted essentially non-oscillatory numerical scheme is the computationally expensive part of the code. The flux computation starts with i) computing the average state, the left and right eigenvectors and eigenvalues of the Jacobian at the average state, ii) projecting the conservative fields and fluxes onto the local characteristic fields using the left eigenvectors matrix, iii) computing the left and right fluxes in characteristic field, and iv) projecting back the numerical fluxes in the physical space using the right eigenvector matrix. These computations are performed in a loop that is ideal for shared memory parallelism. In order to do that, we use the hybrid parallel model with MPI and OpenMP, where MPI is used for internode communication to pass states and interface data from one processor to another, and OpenMP is used for intranode communication to distribute the work equally to each thread. With the hybrid model, a performance improvement on the coarse grid is observed, and the total time to solution is reduced by 20%. However, the pure MPI implementation shows the best scalability on the medium grid on Blue Waters. The good weak and strong scalability of the pure MPI model is because of the optimized work distribution between processes. The problems with OpenMP performance could be due to the memory access and cache use. The use of "numactl", the core layout, plays an important role to achieve scalability. To avoid the bottlenecks with memory and cache, the task placement to distribute MPI processes and threads per processes will be investigated in the future.

ACKNOWLEDGMENTS

This work was supported by a grant from the Shodor Education Foundation through the Blue Waters Student Internship Program. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

REFERENCES

[1] W. Bo, X. Liu, J. Glimm, and X. Li. 2011. A robust front tracking method: Verification and application to simulation of the primary breakup of a liquid jet. *SIAM J. Sci. Comput.* 33 (2011), 1505–1524.

[2] M. Germano, U. Piomelli, P. Moin, and W. H. Cabot. 1991. A dynamic subgrid scale eddy viscosity model. *Phys. Fluids A* 3 (1991), 1760–1765.

[3] J. Glimm, B. Cheng, D. H. Sharp, and T. Kaman. 2020. A crisis for the verification and validation of turbulence simulations. *Physica D: Nonlinear Phenomena* 404 (2020), 132346. https://doi.org/10.1016/j.physd.2020.132346

[4] J. Glimm, H. Kirk, X. L. Li, J. Pinezich, R. Samulyak, and N. Simos. 2000. Simulation of 3D fluid jets with application to the Muon Collider target design. In *Advances in Fluid Mechanics III*, M. Rahman and C. A. Brebbia (Eds.). Vol. 26. WIT Press, Southampton, Boston, 191–200.

[5] J. Glimm, D. H. Sharp, T. Kaman, and H. Lim. 2013. New Directions for Rayleigh-Taylor Mixing. *Phil. Trans. R. Soc. A* 371 (2013), 20120183. https://doi.org/10.1098/rsta.2012.0183 Los Alamos National Laboratory Preprint LA-UR 11-00423 and Stony Brook University Preprint SUNYSB-AMS-11-01.

[6] Torsten Hoefler, James Dinan, Darius Buntinas, Pavan Balaji, Brian Barrett, Ron Brightwell, William D Gropp, Laxmikant V Kale, and Rajeev Thakur. 2013. MPI + MPI: A new hybrid approach to parallel programming with MPI plus shared memory. *Computing (Vienna/New York)* 95, 12 (2013), 1121–1136. https://doi.org/10.1007/s00607-013-0324-2

[7] Ya-Ting Huang and J. Glimm. 2017. A Novel methodology of stochastic short term forecasting of cloud boundaries. *J. Uncertainty Quantification* 5 (2017), 1279–1294.

[8] G. Jiang and C.-W. Shu. 1996. Efficient Implementation of Weighted ENO schemes. *J. Comput. Phys.* 126 (1996), 202–228.

[9] T. Kaman. 2019. Model calibration for turbulent mixing simulations. In *The 16th International Workshop on the Physics of Compressible Turbulent Mixing*, I Houas G. Jourdan and C. Mariani (Eds.). IUSTI UMR 7343, 129–132.

[10] T. Kaman, J. Glimm, and D. H. Sharp. 2010. Initial Conditions for Turbulent Mixing Simulations. *Condensed Matter Physics* 13 (2010), 43401. Stony Brook University Preprint number SUNYSB-AMS-10-03 and Los Alamos National Laboratory Preprint number LA-UR 10-03424.

[11] T. Kaman, R Kaufman, J. Glimm, and D Sharp. 2012. Uncertainty Quantification for Turbulent Mixing Flows: Rayleigh-Taylor Instability. *IFIP Advances in Information and Communication Technology* 377 (01 2012). https://doi.org/10.1007/978-3-642-32677-6_14

[12] T. Kaman, H. Lim, Y. Yu, D. Wang, Y. Hu, J.-D. Kim, Y. Li, L. Wu, J. Glimm, X. Jiao, X.-L. Li, and R. Samulyak. 2011. A Numerical Method for the Simulation of Turbulent Mixing and its Basis in Mathematical Theory. In *Lecture Notes on Numerical Methods for Hyperbolic Equations: Theory and Applications: Short Course Book*. CRC/Balkema, London, 105–129. Stony Brook University Preprint SUNYSB-AMS-11-02.

[13] R. Kaufman, H. Lim, and J. Glimm. 2016. Conservative front tracking: the algorithm, the rationale and the API. *Bulletin of the Institute of Mathematics, Academia Sinica New Series* 11 (2016), 115–130. Stony Brook University Preprint SUNYSB-AMS-15-01.

[14] Parviz Moin and Krishnan Mahesh. 1998. DIRECT NUMERICAL SIMULATION: A Tool in Turbulence Research. *Annual Review of Fluid Mechanics* 30, 1 (1998), 539–578. https://doi.org/10.1146/annurev.fluid.30.1.539 arXiv:https://doi.org/10.1146/annurev.fluid.30.1.539

[15] University of Illinois at Urbana-Champaign National Center for Supercomputing Applications. 2019. Blue Waters Blue Waters. (2019). https://bluewaters.ncsa.illinois.edu

[16] Department of Computer and University of Oregon Advanced Computing Laboratory Information Science. 2020. TAU User Guide. (2020). https://www.cs.uoregon.edu/research/tau/docs/newguide/bk01.html

[17] Extreme Science and Engineering Discovery Environment. 2020. XSEDE. (2020). http://computationalscience.org/xsede-empower

[18] Sameer S. Shende and Allen D. Malony. 2006. The Tau Parallel Performance System. *The International Journal of High Performance Computing Applications* 20, 2 (2006), 287–311. https://doi.org/10.1177/1094342006064482 arXiv:https://doi.org/10.1177/1094342006064482

[19] Chi-Wang Shu. 1998. Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws. In *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations*, A. Quarteroni (Ed.). Lecture Notes in Mathematics, Vol. 1697. Springer, 325–432.

[20] Chi-Wang Shu. 2003. High-order Finite Difference and Finite Volume WENO Schemes and Discontinuous Galerkin Methods for CFD. *International Journal of Computational Fluid Dynamics* 17, 2 (2003), 107–118. https://doi.org/10.1080/1061856031000104851 arXiv:https://doi.org/10.1080/1061856031000104851

[21] J. Smagorinsky. 1963. General Circulation Experiments with the Primitive Equations. *Mon. Weather Rev.* 91 (1963), 99–165.

[22] TOP500.org. 2019. TOP 500 list. (2019).

[23] Shuai Xue. 2015. *A Sharp Boundary Model for Electrocardiac Simulations*. Ph.D. Dissertation. State Univ. of New York at Stony Brook.

[24] H. Zhang, T. Kaman, D. She, B. Cheng, J. Glimm, and D. H. Sharp. 2018. V&V for turbulent mixing in the intermediate asymptotic regime. *Pure and Applied Mathematics Quarterly* 14 (2018), 193–222. Los Alamos National Laboratory preprint LA-UR-18-22134.

[25] Y. Zhou. 2017. Rayleigh-Taylor and Richtmyer-Meshkov instability induced flow, turbulence, and mixing I. *Physics Reports* 720–722 (2017), 1–136.

[26] Y. Zhou. 2017. Rayleigh-Taylor and Richtmyer-Meshkov instability induced flow, turbulence, and mixing II. *Physics Reports* 723–725 (2017), 1–160.

January 2021

Volume 12 Issue 1

ISSN 2153-4136 (online)