Predicting Success in Undergraduate Parallel Programming via Probabilistic Causality Analysis

Sunny Raj and Sumit Kumar Jha
Computer Science Department
University of Central Florida
Orlando, FL, 32816-2362
Email: {sraj, jha}@cs.ucf.edu

***Abstract*—We employ probabilistic causality analysis to study the performance data of 301 students from the upper-level undergraduate parallel programming class at the University of Central Florida. To our surprise, we discover that good performance in our lower-level undergraduate programming CS-1 and CS-II classes is not a significant causal factor that contributed to good performance in our parallel programming class. On the other hand, good performance in systems classes like Operating Systems, Information Security, Computer Architecture, Object Oriented Software and Systems Software coupled with good performance in theoretical classes like Introduction to Discrete Structures, Artificial Intelligence and Discrete Structures-II are strong indicators of good performance in our upper-level undergraduate parallel programming class. We believe that such causal analysis may be useful in identifying whether parallel and distributed computing concepts have effectively penetrated the lower-level computer science classes at an institution.**

***Index Terms*—causality; data analytics; education; predictors; parallel programming;**

I. Introduction

Many application-level programmers have been immune to revolutions in computer architecture and shifts in device technologies for the last five decades. However, the end of Dennard scaling and the consequent rise of dark silicon has ended the exponential growth in processor clock speeds that enabled the same program to run faster on newer generations of computers. In order to compensate for this stagnant processor speed, chip manufacturers have designed general-purpose processors with tens of cores and system builders have put together computers with multiple processors. The consequent rise in the availability of parallel computers now demands that parallel computing should not just be a specialized graduate course; instead, parallel programming ought to be a required skill for all computer science undergraduate students.

Teaching parallel programming to undergraduate students requires a deep understanding of the concepts that must be taught before parallel programming is introduced to them. There have been careful qualitative investigations through pilot studies introducing some parallel programming concepts to core undergraduate CS classes [1–5] and possible strategies for introducing parallel programming to undergraduates have been carefully crafted by expert committees [6–9]. Building upon these foundational studies on parallel computing education, we ask the following question:

- Does success in certain computer science classes cause students to be more likely in succeeding at our undergraduate parallel programming class?

Our results are based on a quantitative *probabilistic causality analysis* of academic performance data of about 301 students. In particular, we observe that success in systems courses such as information security, operating systems, object-oriented software, artificial intelligence and computer architecture leads to good performance in our parallel programming class. Similarly, a strong performance in courses reflecting mathematical maturity such as topics in computer science, problem solving techniques, introduction to discrete structures, and discrete structures II leads to good performance in our undergraduate parallel programming class.

We seek to explain the outcome of our quantitative causal analysis by identifying parallel computing concepts covered in core computer science classes from the computer science body of knowledge and mapping them to exemplar courses in the ACM Computer Science Curricula 2013 [10].

Our work may be used to empirically verify if early adoption efforts at institutions have caused parallel programming concepts to be embedded into introductory classes such as CS-I and CS-II. Unlike validation approaches based on pre/post-surveys and perusal of class syllabuses, our approach can provide long-term third-party evaluation of the outcome of early adoption efforts.

II. Related Work

A. Predicting Success in Courses

Predicting student performance using quantitative methods has been a subject of active interest for at least the last six decades. In 1965, David Lavin put together a one-volume summary [11] of the research in the area between 1953 and 1961. More recently, Willingham et al. have studied the impact of SAT scores and high school GPAs on student performance in college courses [12]. More recently, Belfield and Crosta have shown that high-school GPA have a strong association with college GPAs but other components of the high-school transcript like number of Fs or number of Maths courses do not have a strong impact on college GPAs [13].

Predictive factors for success in computer science classes have also been investigated. Success in introductory CS-I class has been attributed to three factors including comfort level and mathematical background [14]. In another study, it has been shown that SAT scores, high-school rank, and high-school science/maths are determining factors for success as a freshman computer science major [15]. A more recent study has confirmed the importance of Calculus-I, Calculus-II and CS-0 in predicting student's success in the CS-I class [16].

To the best of our knowledge, we are the *first* to quantitatively investigate the impact of other courses on success in the undergraduate parallel programming class.

B. Success in Parallel Programming Courses

A number of qualitative advisory and experience reports on effectively teaching parallel programming to undergraduates have been created due to the success of the recent NSF/TCPP Early Adopter Program [6]. Introduction of parallel and distributed computing concepts in core computer science classes has been validated using statistical analysis of pre/post surveys and performance in examinations [17]. Parallel and Sequential Data Structures and Algorithms now serves as an introductory algorithms course at Carnegie Mellon [18], where sequential programming is only taught as a special case of parallel programming where number of processors is one. Strategies and modules for introducing concurrency in undergraduate classes have been extensively documented [2, 19–24].

Our methodology can be viewed as a validation mechanism that can be used to test successful integration of parallel and distributed computing concepts into core CS classes. For example, our analysis shows that success in CS-1 and CS-2 is not a *prima facie* cause for success in parallel programming. The contents and examinations of the CS-1 and CS-2 course at the University of Central Florida do not significantly include parallel programming concepts. We conjecture that a tight integration of parallel programming in core computer science classes can be quantitatively validated by performing our causality analysis on grades obtained over a period of time and verifying that core CS classes have a strong causal relationship with success in undergraduate parallel programming class.

III. Approach

A. Data

Our data set consists of grades obtained by 301 students in about 15 courses each – producing a total of 12, 937 different grades. The distribution of the grades in the undergraduate parallel programming class is shown in Figure 1. About half the class has obtained A or A- in the class and the rest of the class has obtained B+ or lower grades. More than 13% of the students enrolled in the class obtained a failing grade or withdrew from the class. We are interested in understanding the background of students who do well in the class obtaining an A or A- grade, and relate this to performance in other computer science classes.

The data also includes grades for other computer science (CS) electives, CS core classes and other classes outside the computer science department. Computer science elective classes taken by our students include CIS3362 Cryptography and Information Security, COP4516 Problem Solving Techniques, CAP4630 Artificial Intelligence, CAP4720 Computer Graphics, CNT4714 Enterprise Computing, CIS 3360 Security in Computing, CAP4453 Robot Vision, COP4710 Database Systems, CAP4104 Human and Technology Interaction, COP4020 Programming Languages I, EEL4768 Computer Architecture, COP4600 Operating Systems, CIS3362 Cryptography and Information Security, COT4810 Topics in Computer Science and CAP4053 AI for Game Programming.

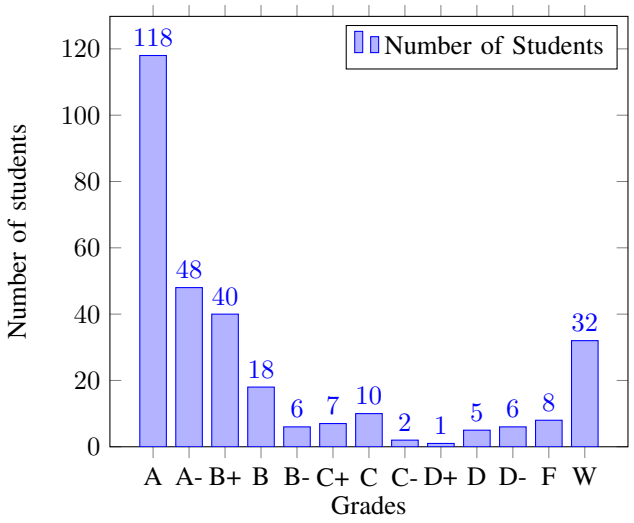

Figure 1. Grade Distribution in Undergraduate Parallel Programming Class

B. Causality

Since the ancient Hindus' quest for a notion of causality [25, p. 73] that suggested effect to have existed in the cause, a lot of progress has been made in determining a more practical definition of causality. Among the researchers who wrote in English, Hume's definition of causality [26] is a good candidate. Hume argues that "repetition of any particular act or operation" leads to the formation of a causal process that "makes us expect for the future, a similar train of events with those which have appeared in the past." For example, observing fire and smoke in constant conjunction leads us to form the causal impression between fire and smoke.

Kant adds a temporal flavor to Hume's idea of causality as constant conjunction of events and requires that the causal event must occur before the effect [27]. According to Kant, "the succession is necessary" and further "the effect . . . follows from it (the cause)." We employ constant conjunction with temporal ordering as our definition of causality in this paper.

C. Probabilistic Causality

Hume's idea of constant conjunction and Kant's notion of temporal ordering work well for natural or scientific facts. For example, consider the observation: "when the sun shines on a rock for more than five hours, the rock gets hot." Since the

Table I
CONDITIONAL PROBABILITY OF OBTAINING AN ABOVE-AVERAGE SCORE IN PARALLEL PROGRAMMING COURSE GIVEN A STUDENT HAS SCORED AN ABOVE-AVERAGE SCORE IN A DIFFERENT COURSE.

Course ID	Course Name	Conditional Probability $P(E^+\|C^+)$ of Good Performance in COP4520	Confidence Score $P(E^+\|C^+) - P(E^+\|\neg C^+)$
COT4810	TOPICS IN COMPUTER SCIENCE	0.877551	0.389456
CIS3362	CRYPTOGRAPHY AND INFO SECURITY	0.909091	0.37116
COP4516	PROB SOLVING TECH & TEAM DYN	0.833333	0.293541
COP4600	OPERATING SYSTEMS	0.722222	0.243549
COP4331	PROC OBJECT ORIENTED SOFTWARE	0.701754	0.241861
EEL4768	COMPUTER ARCHITECTURE	0.71	0.237363
CAP4630	ARTIFICIAL INTELLIGENCE	0.756098	0.236867
COT3100	INTRO TO DISCRETE STRUCTURES	0.616505	0.205979
COP3402	SYSTEMS SOFTWARE	0.621212	0.203736
CGS2545	DATABASE CONCEPTS	0.75	0.201178
COT4210	DISCRETE STRUCTURES II	0.669355	0.200428
CAP4720	COMPUTER GRAPHICS	0.727273	0.189638
COP3223	INTRO TO PROGRAMMING WITH C	0.643357	0.175002
COP3502	COMPUTER SCIENCE I	0.642336	0.166726
CNT4714	ENTERPRISE COMPUTING	0.714286	0.166667
COP4020	PROGRAMMING LANGUAGES I	0.65	0.163812
COP3330	OBJECT ORIENTED PROGRAMMING	0.617647	0.151998
COP3503	COMPUTER SCIENCE II	0.625	0.14849
CIS3360	SECURITY IN COMPUTING	0.634146	0.139764
CAP4453	ROBOT VISION	0.655172	0.114731
CDA3103	COMPUTER LOGIC AND ORGANIZATION	0.575419	0.0590256
COP4710	DATABASE SYSTEMS	0.580645	0.0367121

sun shining for a long time on a rock always makes it hot, Kant's idea of causality is enough to capture the cause (sun shines on a rock for more than five hours) leading to an effect (the rock gets hot).

However, observations in social sciences or measurements on stochastic systems cannot really benefit from the deterministic definitions of Hume or Kant. Consider the observation: "people with SAT scores above 1200 graduate successfully from CS undergraduate programs." While the observation is true for a vast majority of students, there do exist people with SAT scores above 1200 who do not graduate from CS undergraduate programs. Suppes' theory of probabilistic causation is a potential solution to this problem, Given an event E and another event C, C is said to be a *prima facie* cause of the event E only if $P(E|C) > P(E)$.

In this paper, we use this probabilistic notion of causality and call C to be a cause for E if event C raises the probability of the event E and the event C occurs before event E.

IV. RESULTS

Course performance data from 301 students was partitioned into three groups: *above-average*, *below-average* and *withdrawn*. The top 50^{th} percentile of students who completed a course were assigned into the *above-average* group and the remaining students who completed the course were assigned to the *below-average* group. Those students who registered for the course but did not finish the course were assigned to the *withdrawn* group. For the parallel programming course, students obtaining grades A and $A-$ were assigned to the *above-average* group, students with grade W were assigned to the withdrawn group, and the rest were put into the *below-average* group. These three groups for each course were treated as individual events during causal analysis. In the context of our empirical investigations, a student being classified in the *above-average* group for a course corresponds to the student doing well in that course.

A. Probabilistic Causal Analysis

We compute the impact of a student's performance in individual courses on performance in the parallel programming course. We use E^+ to denote the effect that a student has scored above average in the parallel programming class. Similarly, we use the notations C^+ to denote that a student has scored above-average in a course C. The notation C^- captures the event that a student has not scored above-average in the course. The scenario that a student dropped out from the course is captured by the notation C^0. The impact of a student's performance in course C on the parallel programming class is obtained by computing the difference of the probability of $P\big(E^+|C^+\big)$ and the probability $P\big(E^+|(C^- \cup C^0)\big)$.

Table I shows the top 22 classes that have a positive impact on the outcome of our undergraduate parallel programming class. The first column of the table contains the ID of the courses, the second column is the course name, and the third column denotes the conditional probability that an above-average score in a course will lead to an above-average score in the parallel programming class. The rightmost column in the table is our probabilistic confidence in a course having a causal impact on the parallel programming course.

Ideally, one would like to obtain a temporal sequence of courses that leads to strong performance in our undergraduate parallel programming class. However, our data set is limited to 301 students and different students have pursued sufficiently different trajectories during their undergraduate coursework to

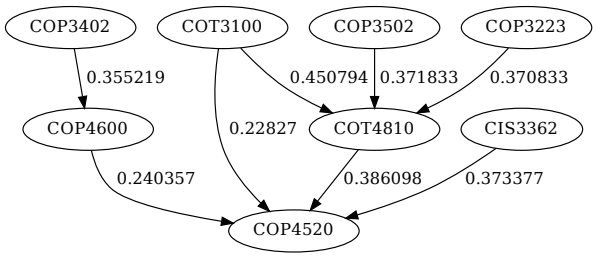

Figure 2. Causal Network showing confidence score $\left(P(E^+|C^+) - P(E^+|\neg C^+)\right)$ relationships between courses such that performing above-average on one course leads to an above-average performance in another course.

make the determination of such an optimal sequence of courses infeasible. Hence, we have relied on the pairwise causality analysis between performance in individual courses and our undergraduate parallel programming class.

B. Causal Graph

We create a Suppes-Bayes causal network [28] to understand the causal relationship from different courses to the undergraduate parallel programming course. The first layer of the network consists of 2000 and 3000-level courses. These courses are pre-requisites for the upper-level 4000-level courses. The second layer consists of advanced 4000-level courses that a typical student takes before enrolling for our undergraduate parallel programming course. The third layer consists of a single node representing the parallel programming class. The course progression of few students is atypical; there are a small number of students that take a 4000-level course before a 3000-level course. We do not consider these atypical cases for our computation of causal relationships. For any given directed edge from node C to node E in the Suppes-Bayes causal network, we only consider those instances of data when the course C is taken before the course E. This does not significantly alter the size of our data set.

We compute the probabilities $P(E^+|C^+)$ and $P(E^+|\neg C^+)$ for every course C and the parallel algorithms course E as long as the course C occurs before the course E i.e. the pair of courses follow the temporal ordering required for causality. The pair of nodes corresponding to courses having a positive value of $P(E^+|C^+) - P(E^+|\neg C^+)$ are connected by an edge to create a Suppes-Bayes causal network.

The calculation of $P(E^+|\neg C^+)$ for the causal network differs slightly from the results in Table I as we restrict our consideration only to those instances of the data that are present at that level of the network.

Full causal network containing causality relationship between all the nodes is not readily comprehensible to a human expert. We use the Bayesian Information Criterion (BIC) [29] to synthesize a subgraph of the original network that optimizes the structural complexity of the network against the likelihood of the data set. In our experiments, we initially create a random subgraph of the causal network. Then, we perform a hill-climbing search in the neighborhood of the graph such that the Bayesian Information Criterion (BIC) increases. We stop the execution of our algorithm after $100,000$ iterations or when there is no increase in the Bayesian Information Criterion.

The result of our analysis is shown in Figure 2. The courses that directly impact student performance include systems courses such as COP4600 Operating Systems and CIS3362 Crytopgraphy and Information Security as well as theoretical courses such as COT3100 Introduction to Discrete Structures and COT4810 Topics in Computer Science.

C. Predicting Student Grades

We employ machine learning and insights gained from probability causality analysis to predict student grades in our parallel programming course. For grade prediction, we use performance data from 10 courses having the highest confidence scores from table I. We use support vector classification implemented in the popular machine learning toolkit *scikit-learn* to predict *above-average* or *below-average* grades of students [30]. Even though there are a total of 301 students taking parallel programming course, the information contained in each of these student profiles is only partial as most student only take a subset of the 10 courses that we are using for prediction. Hence, we choose the Leave One Out Cross Validation (LOOCV) scheme for analyzing the performance of our support vector classifier. We train the support vector machine on all data points except one and then use the trained algorithm to predict the outcome of the left-out data point. The training algorithm is run n times where n is equal to the number of data-points and the average of the n performances is used as an overall measure of the performance of the prediction system. We were able to make the correct predictions about student grades 65.33% of times. This prediction was obtained using a radial basis kernel function with $C = 13$ and $\gamma = 0.1$ [31]. On training the machine learning algorithm using the full data set, we were able to get correct predictions for 85.33% of cases. This performance was obtained using a polynomial function with degree 3 and $C = 1000$. The results of this experiment are shown in Figure 3.

D. Qualitative Analysis of Results

The Body of Knowledge (BoK) in the ACM Computer Science Curricula 2013 [10] requires 5 core-tier-1 hours and 10 core-tier-2 hours of parallel and distributed computing. The core topics covered by the ACM BoK include the following:

1) Goals of parallelism vs. concurrency
2) Synchronization constructs
3) Data races and higher-level races
4) Independence and Partitioning
5) Threads, SIMD and MapReduce
6) Shared memory and consistency
7) Message passing

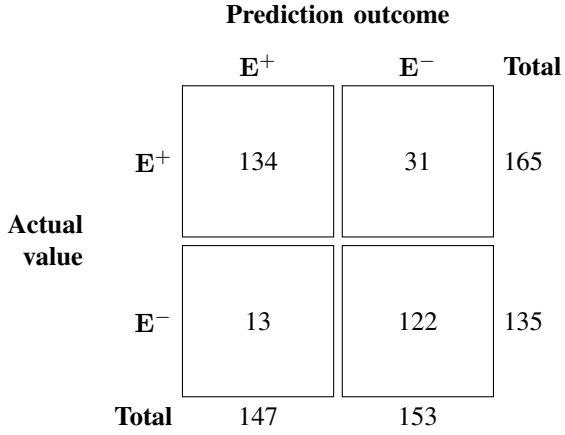

Figure 3. Confusion matrix showing the results of prediction of student grades using a classifier trained on the whole dataset.

8) Atomicity and mutual exclusion using locks, semaphores and monitors
9) Critical paths, work, span and Amdahl's law
10) Speed-up and scalability
11) Embarrassingly parallel algorithms, patterns (divide & conquer, map & reduce, master-workers)
12) Multicore processors
13) Shared vs. distributed memory
14) SIMD, SMP, and vector processing

Courses other than the upper-level parallel programming elective should cover these core topics. We analyze the exemplar syllabi included in the ACM Computer Science Curricula 2013 to identify the number of topics covered by different courses.

Our qualitative analysis shows that systems courses such as operating systems and computer architecture build the foundations required to study an upper-level on parallel programming. On the other hand, exemplar syllabuses of courses such as CS-1 and CS-2 in the ACM Computer Science Curricula 2013 do not yet share a significant overlap with the foundations of parallel programming. Thus, our qualitative analysis supports our quantitative findings that success in parallel programming courses relies on good performance in upper-level systems courses.

The dependence of above-average performance in parallel programming on theoretical courses like discrete structures may be explained by improved mathematical maturity and a deeper understanding of different computing models such as finite-state automata, push-down automata and Turing machines.

V. Conclusions

Our empirical analysis shows that success in certain systems and theoretical computer science classes indeed has a strong effect on good performance in our upper-level undergraduate parallel programming class. Further, the inadequacy of parallel programming concepts in our CS-1 and CS-2 classes is reflected in our empricial results as good performance in these classes does not serve as a *prima facie* cause for good performance in our upper-level parallel computing class.

Our causality analysis approach may be used to validate the efficacy of early adoption efforts. A successful early adoption effort can be identified by high causal probabilities linking low-level courses to the upper-level undergraduate parallel programming course.

Our analysis is based on data obtained from an upper-level undergraduate parallel programming class at the University of Central Florida (UCF). Deeper and possibly different insights may be obtained by analyzing longitudinal student data from different types of institutions at various stages of their early adoption effort.

Acknowledgement

The work is supported by the University of Central Florida Predictive Analytics Innovation Fellow Program. The authors would like to thank the US Air Force for support provided through the AFOSR Young Investigator Award to Sumit Jha. The authors acknowledge support from the National Science Foundation Software & Hardware Foundations #1438989 and Exploiting Parallelism & Scalability #1422257 projects.

References

[1] Michael Wrinn. "Parallel computing: thoughts following a four-year tour of academic outreach". In: *ACM Inroads* 3.3 (2012), pp. 4–8.

[2] Steven Bogaerts and Joshua Stough. "Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses". In: *NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing* (2013).

[3] Clayton Ferner, Barry Wilkinson, and Barbara Heath. "Using patterns to teach parallel computing". In: *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*. IEEE. 2014, pp. 1106–1113.

[4] Suzanne J Matthews. "Using Phoenix++ MapReduce to introduce undergraduate students to parallel computing". In: *Journal of Computing Sciences in Colleges* 32.6 (2017), pp. 165–174.

[5] Steven Bogaerts, Kyle Burke, and Eric Stahlberg. "Integrating parallel and distributed computing into undergraduate courses at all levels". In: *First NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar-11), Anchorage, AK*. 2011.

[6] Sushil K Prasad et al. *Topics in parallel and distributed computing: introducing concurrency in undergraduate courses*. Morgan Kaufmann, 2015.

[7] Sushil K Prasad et al. "Literacy for all in parallel and distributed computing: guidelines for an undergraduate core curriculum". In: *CSI Journal of Computing* 1.2 (2012), pp. 82–95.

[8] Joel Adams, Richard Brown, and Elizabeth Shoop. "Patterns and exemplars: Compelling strategies for teaching parallel and distributed computing to cs undergraduates". In: *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*. IEEE. 2013, pp. 1244–1251.

[9] Sushil K Prasad et al. "NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing: core topics for undergraduates". In: *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM. 2014, pp. 735–735.

[10] Mehran Sahami et al. "ACM/IEEE-CS computer science curriculum 2013: reviewing the ironman report". In: *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM. 2013, pp. 13–14.

[11] David E. Lavin. *The Prediction of Academic Performance: A Theoretical Analysis and Review of Research*. Russell Sage Foundation, 1965, p. 182.

[12] Warren W Willingham et al. *Predicting college grades: An analysis of institutional trends over two decades.* Educational Testing Service, 1990.

[13] Clive R Belfield and Peter M Crosta. "Predicting Success in College: The Importance of Placement Tests and High School Transcripts. CCRC Working Paper No. 42." In: *Community College Research Center, Columbia University* (2012).

[14] Brenda Cantwell Wilson and Sharon Shrock. "Contributing to success in an introductory computer science course: a study of twelve factors". In: *ACM SIGCSE Bulletin*. Vol. 33. 1. ACM. 2001, pp. 184–188.

[15] Patricia F Campbell and George P McCabe. "Predicting the success of freshmen in a computer science major". In: *Communications of the ACM* 27.11 (1984), pp. 1108–1113.

[16] Lynn Lambert. "Factors that predict success in CS1". In: *Journal of Computing Sciences in Colleges* 31.2 (2015), pp. 165–171.

[17] Sandeep Kumar. "oriented teaching of PDC topics in integration with other undergraduate courses at multiple levels: A multi-year report". In: *Journal of Parallel and Distributed Computing* 105 (2017), pp. 92–104.

[18] Randal E Bryant. "EduPar 2016 Keynote". In: *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*. IEEE. 2016, pp. 941–941.

[19] Thomas H Cormen. "Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses". In: *NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing* (2013).

[20] Dan Grossman. "Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses". In: *NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing* (2013).

[21] Ramachandran Vaidyanathan, Jerry L Trahan, and Suresh Rai. "Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses". In: *NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing* (2013).

[22] David P Bunde. "Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses". In: *NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing* (2013).

[23] Ryan E Grant and Stephen L Olivier. "Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses". In: *NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing* (2013).

[24] Victor Eijkhout. "Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses". In: *NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing* (2013).

[25] Theos Bernard. *Hindu philosophy*. Greenwood Press, 1947.

[26] David R Shanks. "Hume on the Perception of Causality". In: *Hume Studies* 11.1 (1985), pp. 94–108.

[27] Gerd Buchdahl. "Causality, causal laws and scientific theory in the philosophy of Kant". In: *The British Journal for the Philosophy of Science* 16.63 (1965), pp. 187–208.

[28] Gelin Gao, Bud Mishra, and Daniele Ramazzotti. "Efficient Simulation of Financial Stress Testing Scenarios with Suppes-Bayes Causal Networks". In: *Procedia Computer Science* 108 (2017), pp. 272–284.

[29] Gideon Schwarz. "Estimating the Dimension of a Model". In: *Ann. Statist.* 6.2 (Mar. 1978), pp. 461–464. DOI: 10.1214/aos/1176344136. URL: https://doi.org/10.1214/aos/1176344136.

[30] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[31] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 0387310738.