Introducing parallel programming across the undergraduate curriculum through an interdisciplinary course on computational modeling

Building Big Systems for Biting into Big Data

Narsingh Deo, Faraz Hussain, Sumit Kumar Jha, Mahadevan Vasudevan
Electrical Engineering and Computer Science Department
University of Central Florida
Orlando FL 32816 USA
E-mail: deo,fhussain,jha,maha@eecs.ucf.edu

***Abstract*—In this paper, we argue a case for including an interdisciplinary course on computational modeling with a focus on parallel programming across the undergraduate curriculum. The construction of computational models has become a fundamental process in the discovery process for all scientific disciplines, and there is little instructional support to enable the next generation of scientists and engineers to effectively employ massively parallel high-performance computing machines in their scientific process. We believe that a first course in computer programming must be followed by a second interdisciplinary course in computational modeling with a focus on parallel programming for students across the undergraduate curriculum.**

Keywords-parallel programming; parallel algorithms; numerical methods; discrete algorithms; computational modeling

I. INTRODUCTION

The end of exponential growth [1] in the computing power available on a single processing element has given birth to an era of massively parallel computing where every programmer must be trained in the art and science of parallel programming. While the processing power available on each processing core may not exceed a teraflop in the foreseeable future, we expect exascale parallel machines [2] to be available by the year 2020. This has given rise to an urgent need for scientists that are well trained in the art and science of parallel computer programming. Traditionally, such specialized tasks have been delegated to post-doctoral high-performance computing researchers and computer scientists working in a limited number of specialized parallel programming laboratories. However, it is now clear that the ubiquitous demand for parallel programming will require the training of undergraduate students across multiple disciplines (e.g., physics, chemistry, biology) in this crucial area.

We propose the introduction of parallel programming algorithms for solving challenging but important problems in the discovery, analysis and validation of complex computational models across the undergraduate curriculum. Sitting at the intersection of big data, computational statistics, and model checking, the area of computational modeling provides a good choice for introducing undergraduates to the need and power of parallel computing in a wholesome and natural manner.

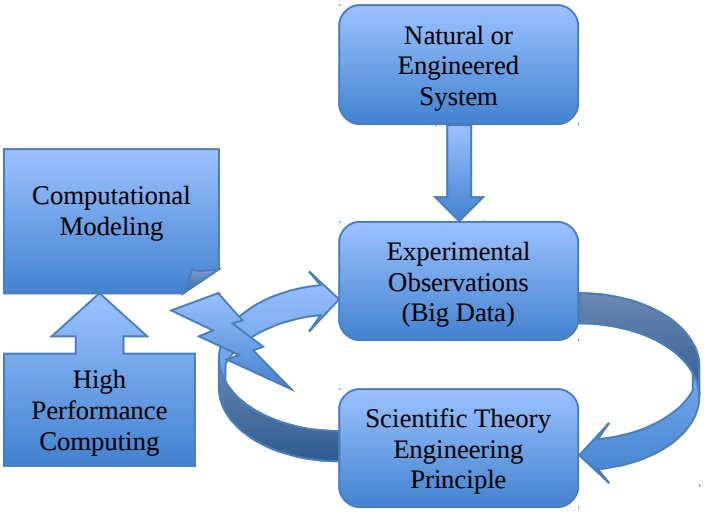

Figure 1: The role of computational modeling and parallel algorithms in sciences and engineering.

In this paper, we will identify various topics, algorithms, and learning outcomes that can be achieved from such a course in computational modeling. As an early adopter of the NSF TCPP program, we endeavor to introduce such an elective into our undergraduate program. The University of Central Florida is the second-largest university in the country, and will provide an ideal test-bed for the course suggested herein. A basic undergraduate course in programming and a course in calculus (currently in place) will serve as the prerequisites for the proposed senior-level undergraduate course for all science and engineering students.

II. RELATED WORK

Our experience as early adopters of parallel programming curriculum in past years and continued collaboration with our industrial affiliates has led to this approach in implementing pedagogical strategies in this proposed interdisciplinary course. While several other approaches for incorporating parallelism in undergraduate curriculum have been proposed over the past two years [3,4], they have addressed programming models in particular. Parallel programming languages, such as MPI, CUDA, and OpenMP, are introduced along with well-defined problems and solutions to enhance the parallel programming skills of computer science students. Instead, our focus is on introducing parallel computational thinking based problem-solving techniques for real-world problems across the entire spectrum of the undergraduate curriculum.

III. LEARNING COMPUTATIONAL MODELS

The nature of available data for building models has changed in two fundamental ways over the last century: (i) petabytes of data are now easily obtained using computer-aided data acquisition systems, and this trend is going to continue in the foreseeable future [5], as well as (ii) the arrival of structured data [6] from sources such as social networks, biochemical networks, and prices of financial instruments. The analysis of exascale data will require algorithms that are fundamentally parallel and can be deployed on upcoming exascale hardware. On the other hand, the arrival of structured big data requires the creation of new parallel algorithms that are not simply an extension of traditional machine learning approaches employed for analyzing unstructured time series data.

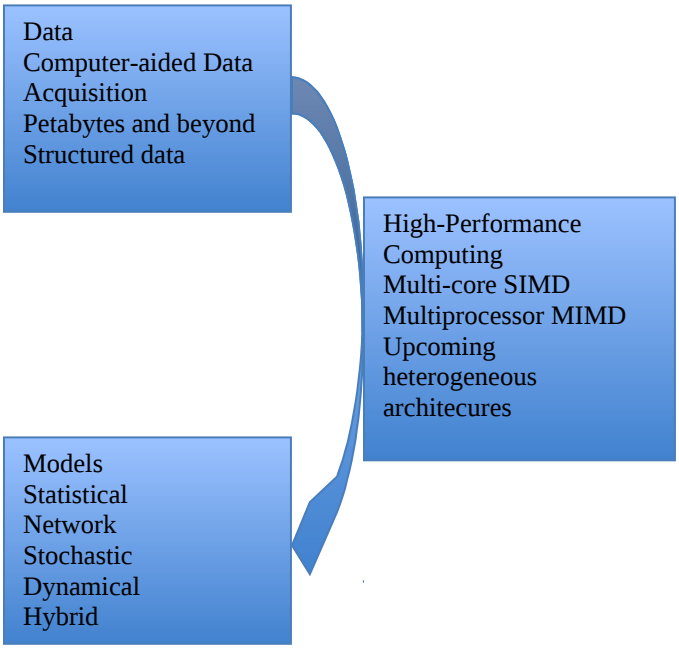

Figure 2: Transforming data into knowledge using HPC

A. *Learning statistical models from big data*

The success of machine learning has brought smart recommendation systems into our daily lives – news websites, online music streaming, web ads, online movie rentals, and even email services use these recommendation systems to tailor their online presence to our own personal choices. Undergraduate students across the curriculum have had substantial experience with these recommendation systems and can easily relate to algorithms that seek to develop such recommendation systems from extreme scale data.

In particular, we propose the teaching of parallel algorithms for learning decision trees from large data sets. A decision tree is a hierarchical data structure that is built using recursive partitioning. Each internal node in a decision tree is obtained by dividing the available data set into two or more pieces – each assigned to the children of this internal node. A fundamental challenge in learning decision trees is the appropriate choice of the attribute and the split points that separate the data into multiple fragments.

Parallel algorithms for constructing decision trees serve as good prototypical examples for distributed-memory parallel-data algorithms. In one approach [7], the data set is distributed equally among multiple slave nodes. A master node seeks to determine the optimal attributes and splitting strategies at each internal node. The slave nodes assist the master node by computing local class frequencies for categorical attributed, and by sorting the numerical attributed and exchanging split points with other slave nodes. Besides serving as a prototypical example for traditional master-slave parallel algorithms architecture, the example will reemphasize the computational thinking approach among undergraduate students.

B. *Learning dynamical graph models from structured data*

The dramatic growth of the World Wide Web (Web) and the lnternet has brought the study of large, random real-life networks into prominence. Concepts like social networks are intuitively familiar to most of our students across the undergraduate curriculum. Empirical studies have found statistical similarities between the WWW and Internet networks and many other real-life networks, such as the network of phone calls, genetic regulatory networks, metabolic networks, neural networks, and various infrastructure networks. Viewed as large, random graphs, in which birth and death of nodes and edges are taking place, these networks already enjoy a truly cross-disciplinary following (involving mathematics, statistics, physics, biology, engineering, and computer science) aimed at understanding the fundamental properties and functions of various natural and engineered systems. Thus, we believe that parallel algorithms for learning such graph models from structured data will provide a suitable avenue for reaching a large cross-section of undergraduate students.

An iterative interplay between statistical analysis of real-world networks and probabilistic modeling will be used to develop dynamical models of such complex networks [8, 9]. The students will be faced with the challenge of dividing structured data into multiple processors for computational

efficiency, and will be exposed to the design decisions for implementing parallel algorithms that must be made depending on the nature of the network being analyzed. Topological properties of the network, such as graph diameter, k-separators, edge density, number of clusters, will be used to construct suitable partitions of the data among multiple processing elements.

C. Learning parameters of computational models

The success of high-performance computing has facilitated the rapid development of increasingly complex models of natural and engineered systems by students in biology, physics, chemistry, and even finance. While the development of such models requires considerable domain knowledge and arguably little knowledge of the science of computing itself, a key problem in computational modeling that cuts across boundaries of scientific disciplines and motivates the development of new massively parallel high-performance algorithms is *the discovery of parameters for computational models [10]*.

The overall structure of computational models can often be obtained from first principles by using discipline-driven understanding and insight into the physical system that is being modeled. However, several components of a computational model are not readily obtained from first principles. Very often, model designers incorporate such information in the model as *parameters*. The model designer chooses these parameter values carefully so that the computational model replicates the behavior of the natural or engineered system being modeled. Thus, a key problem in computational modeling is the identification of such parameters.

Undergraduate students already work with models of varying complexity as a part of their course work in computer architecture, electrical circuits, biochemistry, control theory, physics, pricing derivatives, etc. In this module, we will introduce them to the inverse problem of parameter synthesis and demonstrate the power of even simple parallelism in solving the parameter-synthesis problems. We will then present parallel variants of filtering algorithms for learning parameters of computational models. We will also study the challenges involved in distributing parameter discovery methods such as simulated annealing and evolutionary algorithms. The focus of the discussion will be on the design space of parallel algorithms and the various choices available to a scientist while transforming sequential algorithms into their parallel cousins.

D. Learning communities in complex evolving networks

Community detection [11, 12] is a branch of network science that deals with the characterization, definition, extraction and identification of close-knit nodes in a complex network. Our undergraduate students are already familiar with the intuitive notion of communities from online social networks, and the teaching of massively parallel community detection algorithms will provide a suitable pedagogical framework for introducing students to compute-intensive problems that may benefit from parallelism. Every node in a real-world network model can be associated with a closely-knit community that naturally defines the set of nodes related to the given node in that network. Formally, dense subgraphs in the large sparse graphs are called as communities. The presence of communities in graphs has been studied [11] and algorithms have been investigated to identify communities in static graphs. In many of these problem instances, the immutable social network itself may be replicated on every processing element or reside on a shared memory resource. For example, in the case of community identification, where we are interested in the community to which a given seed node belongs, a large complex network can reside in the shared memory, and the processing elements can act upon this network with different seed nodes to extract relevant communities. The distributed processing and systematic approach tackles the dynamicity of the network.

IV. MODEL SIMULATION

Algorithms for model simulation provide a natural avenue for the use of high performance computing platforms and algorithms. It has been our observation that trivial parallelism by replicating the simulation of a stochastic model among multiple processors is routinely used by our students across the undergraduate curriculum. Because of the simplicity of the approach and the lack of training in more sophisticated parallel programming methodologies, this continues to be used by doctoral students and post-doctoral researchers in non-informatics disciplines. In the proposed course, we will focus on simulation algorithms that enable our students to perform a more fine-grained parallel simulation of computational models.

A. Parallel simulation of ODE models

The use of single-program multiple-data parallelism for accelerating the solution of the initial-value problem has been studied in literature, and is available as readily usable software from a variety of libraries. The numerical solution to the initial value problem for ordinary differential equations(ODE) provides opportunity for parallelism at several levels: (i) ODE systems render themselves to parallel computation of the values of multiple differential variables on shared-memory parallel architectures like CUDA, (ii) sparse ODE systems where variables evolve at multiple scales render themselves to further parallelism based on optimistic look-ahead, (iii) adaptive algorithms where the step-size may need to be computed further benefit from a parallel exploration of the right step size.

The parallel simulation of ODEs has produced two outstanding successes: (i) powerful circuit simulators capable of simulating ASIC designs [13], and (ii) biochemical simulations ranging to whole-cell models [14]. This has happened despite the theoretical challenges involved in parallelizing arbitrarily explicit one-step methods and implicit Runga-Kutta methods. The use of the details of the model to gain insight into the inherently parallel nature of the simulation of circuits and biochemical models, despite the negative theoretical results, will provide an interesting insight for our undergraduate students.

B. Parallel simulation of CTMCs

Several interesting models, including those in computational systems biology and protocol design, are described by continuous time Markov chains (CTMCs). Various stochastic simulation algorithms, including the Gillespie simulation [15], are used to explore the behavior of such stochastic models. Every step of the simulation algorithm is expensive, and requires the generation of two random numbers. Further, each step of the numerical simulation process must be completed before a new state of the model can be generated and the probability distribution of the various events at the next step can be determined correctly. Thus, the stochastic simulation algorithm presents a unique challenge to the problem of parallel simulation.

We will study how approximations to the exact stochastic simulation algorithm can be computed that do not require such a tight sequential step-by-step computation of events [16]. In particular, we will focus on methods that can use the structure of the continuous time Markov chain model to identify two or more sub-models that can be executed in parallel without substantial communication bottleneck. This will be useful in simulating large biochemical reaction systems such as whole-cell models.

C. Parallel simulation of ABMs

Popular frameworks for Agent-Based Models—including NetLogo, Repast, and SPARK--are routinely used to analyze and predict the evolution of complex systems. Several existing agent-based modeling frameworks, including MASON, provide parallel implementations based on shared-memory parallel architecture.

A parallel simulation [17, 18] of an agent-based model must respect the synchronous execution of various agents, and such data coherency requires a careful synchronization of executing parallel processes that represent different agents of an ABM. Static analysis of a given agent based model can be used to identify independence between the evolution of agents, and the consequent parallel but synchronous execution of such agents. This course module will educate the students in the use of algorithm for identifying opportunities for parallelism in simulation of agent based models.

D. Parallel simulation of SDEs

Stochastic differential equations are used in a number of applications, including computational finance, cyber-physical systems, and biochemical modeling. In this module, we will explore the use of GP-GPUs for accelerating the solution of stochastic differential equations. In this context, the students will explore the use of GPUs for efficient generation of random numbers. We will then study the parallelization of Euler and Milstein discretization schemes using multi-core shared-memory and distributed memory architectures.

V. ANALYSIS AND VALIDATION OF MODELS

The validation of computational models against experimental observations and expert insight is an area of active interest across all disciplines that employ computational modeling. In the proposed course, we will cover the topics of statistical and symbolic model checking using high-performance computing. Model checking is an area of computer science that focuses on the validation of computational models against formal specifications encoded in temporal logics. The technique has been applied to design correct-by-construction integrated circuits, device drivers, and other software. There is growing interest in extending the area to more complex quantitative models, such as those used to describe cyber-physical systems.

A. Statistical model checking

A promising area for high performance computing is the use of statistical hypothesis testing in the determination of the correctness of a model with respect to a stated formal specification. Because of the use of sampling methods, the statistical model checking approach is readily parallelized using distributed-memory parallel architectures [19, 20, 21]. The verification of every simulation trajectory against a suitable linear temporal logic fragment has itself been parallelized in a non-trivial manner. Thus, the students will be exposed to both trivial and non-trivial kinds of parallelization through this course module.

B. Symbolic model checking

The use of symbolic data structures like Binary Decision Diagrams, algorithms like the DPLL based satisfiability techniques, and theorem-proving methods have also been used to verify the correctness of computational models against formal specifications. Variants of satisfiability and SMT solvers that can employ the shared-memory parallelization [22] available in GPGPUs will be studied. The challenges involved in keeping the learnt clause database coherent but small in a distributed setting will be explored. We will also study parallel variants of popular symbolic data structures like Binary Decision Diagrams.

VI. INCORPORATING INDUSTRY STANDARD PROGRAMMING

One of the key challenges faced by students transitioning from academia to industry lies in adapting to newer technologies used by modern day web-based, online-centric companies. The demand and the competition have necessitated the usage of a variety of new and effective programming paradigms especially in the parallel programming space that are different from the traditional programming languages taught in school. There is no doubt that learning the right computing methodologies, data structures, programming practice and object-oriented techniques in the early stages of a career is critical. In the proposed course, we will ensure that the students familiarize themselves with state-of-the-art languages.

The advent of cloud-based technologies, web-based solutions, mobile platforms and the demand for social networking applications has increased the usage of distributed computing models and frameworks for processing large data sets. The ubiquitous nature of big data across medical, social and marketing domains continuously creates the opportunity for the usage of distributed processing framework such as MapReduce. This

programming model that has gained significant attention in the recent past especially amongst cloud-based architectures and user-centric net applications such as Twitter, Yelp, etc. is MapReduce [28, 27, 26]. The computational demand dictated by the processing of large data sets has been satisfied only by the usage of multi-core and multi-processor programming. MapReduce facilitates such large data set processing by automatically parallelizing programs written in functional style and executing them on a large cluster of machines. Several improvements for this model have been proposed over the years, and usage has also continued to grow [25, 29, 30]. High scalability and natural blend with multi-core and multi-processor programming are the stand out features of MapReduce.

Hadoop is an open-source implementation of the MapReduce model for reliable, scalable, distributed computing developed and maintained by Apache [24, 31, 32]. Several internet giants such as Yahoo, Google, Amazon, eBay, Netflix, etc. and numerous startups [23] have adapted this powerful framework. The simplistic yet powerful nature of the model with the software libraries that support data-intensive distributed applications makes it a natural choice for day-to-day social networking applications.

VII. CONCLUSION

We have proposed a new interdisciplinary course for teaching parallel computational thinking to senior undergraduate students across the undergraduate curriculum. Our proposed course will provide adequate tools to our future scientists and engineers so that they can leverage advances in high-performance computing to excel in the new and upcoming area of data-driven sciences (See Figure 1). We have also summarized the key modules involved in the design of such a course along with their pedagogical goals in Figure 3. It is our belief that such a broad course that focused on computational modeling will enable a large number of undergraduate students to learn the art and science of high-performance computing in a setting that they are likely to revisit in their professional life as scientists and engineers.

Module	Pedagogical value in parallel programming	Additional teaching goals
Decision Trees	Distributed-memory Master-slave Algorithm	Building a recommendation system from big data via classical machine learning.
Dynamical graph models	Partitioning of structured data in distributed memory applications.	Building models of structured data. Theory of complex network models.
Parameter Synthesis	Trivial parallelism, parallelizing sequential code, thinking in a parallel manner.	Use of evolutionary algorithms, simulated annealing, and optimization methods
Learning communities	Provoking parallel solutions to Graph theoretic algorithms.	Modeling intensive biological data and observing correlation with real-world data
Parallel Simulation	SIMD and use of static analysis methods for discovering parallelism.	Simulation of ODEs, SDEs, ABMs and CTMCs.
Model Validation	Parallel complexity, discrete and numerical parallel algorithms, cache-coherency.	Model checking, symbolic data structures, statistical hypothesis testing.

Figure 3: Proposed modules and their pedagogical goals.

ACKNOWLEDGMENT

The authors would like to acknowledge that the "Parallel Computation in Undergraduate Computer Science Curriculum" at UCF was first developed and implemented under an NSF grant # CDA-9115281 starting in 1991. We also thank the NSF-TCPP Early Adopter program for their more recent and sustained support for several years.

REFERENCES

[1] *Understanding Moore's Law: Four Decades of Innovation.* Edited by David C. Brock. Philadelphia: Chemical Heritage Press, 2006. ISBN 0-941901-41-6. OCLC 66463488

[2] MPI at Exascale: Challenges for Data Structures and Algorithms. Abstract of Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, Volume 5759. ISBN 978-3-642-03769-6. Springer Berlin Heidelberg, 2009, p. 3.

[3] M. ACACIO et al, An Experience of Early Initiation to Parallelism in the Computing Engineering Degree at the University of Murcia, Spain. In Proceedings of IEEE EduPar 2012.

[4] N. GIACAMAN, Teaching by Example: Using Analogies and Live Coding Demonstrations to Teach Parallel Computing Concepts to Undergraduate Students. In Proceedings of IEEE EduPar 2012.

[5] Ranganathan, Parthasarathy. "THE DATA EXPLOSION." (2011).

[6] Chang, Fay, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh,Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. "Bigtable: A distributed storage system for structured data." *ACM Transactions on Computer Systems (TOCS)* 26, no. 2 (2008): 4.

[7] Kufrin, Richard. "Decision trees on parallel processors." *Machine Intelligence and Pattern Recognition* 20 (1997): 279-306.

[8] Albert, Réka, and Albert-László Barabási. "Statistical mechanics of complex networks." *Reviews of modern physics* 74, no. 1 (2002): 47.

[9] Deo, Narsingh, and Aurel Cami. "Preferential deletion in dynamic models of web-like networks." *Information processing letters* 102, no. 4 (2007): 156-162

[10] Tarantola, Albert. *Inverse problem theory and methods for model parameter estimation*. Society for Industrial Mathematics, 2005

[11] Fortunato, Santo. "Community detection in graphs." *Physics Reports* 486, no. 3 (2010): 75-174.

[12] Vasudevan, Mahadevan, Hemant Balakrishnan, and Narsingh Deo.

"Community discovery algorithms: an overview." *Congressus Numerantium* 196 (2009): 127-142.

[13] Krasnicki, Michael, Rodney Phelps, Rob A. Rutenbar, and L. Richard Carley. "MAELSTROM: efficient simulation-based synthesis for custom analog cells." In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pp. 945-950. ACM, 1999.

[14] Takahashi, Kouichi, Katsuyuki Yugi, Kenta Hashimoto, Yohei Yamada, Christopher JF Pickett, and Masaru Tomita. "Computational challenges in cell simulation: a software engineering approach." *Intelligent Systems, IEEE* 17, no. 5 (2002): 64-71.

[15] Ripley, Brian D. *Stochastic simulation*. Vol. 316. Wiley, 2009.

[16] Schwehm, Markus, U. Brinkschulte, K. E. Grosspietsch, C. Hochberger, and E. W. Mayr. "Parallel stochastic simulation of whole-cell models." In *Proceedings of the Second International Conference on Systems Biology (ICSB 2001), Los Angeles, CA*, pp. 4-7. 2001.

[17] Perumalla, Kalyan S., and Brandon G. Aaby. "Data parallel execution challenges and runtime performance of agent simulations on gpus." In *Proceedings of the 2008 Spring simulation multiconference*, pp. 116-123. Society for Computer Simulation International, 2008.

[18] Fujimoto, Richard M. "Parallel discrete event simulation." *Communications of the ACM* 33, no. 10 (1990): 30-53.

[19] Jha, Sumit Kumar, Raj Gautam Dutta, Christopher J. Langmead, Susmit Jha, and Emily Sassano. "Synthesis of insulin pump controllers from safety specifications using Bayesian model validation." *International Journal of Bioinformatics Research and Applications* 8, no. 3 (2012): 263-285.

[20] Jha, Susmit. "Statistical Analysis of Privacy and Anonymity Guarantees in Randomized Security Protocol Implementations." *arXiv preprint arXiv:0906.5110* (2009).

[21] Jha, Sumit, Edmund Clarke, Christopher Langmead, Axel Legay, André Platzer, and Paolo Zuliani. "A bayesian approach to model checking biological systems." In *Computational Methods in Systems Biology*, pp. 218-234. Springer Berlin/Heidelberg, 2009.

[22] Croix, John F., and Sunil P. Khatri. "Introduction to GPU programming for EDA." In *Proceedings of the 2009 International Conference on Computer-Aided Design*, pp. 276-280. ACM, 2009.

[23] K. BOGUTA, *Hadoop & Startups* http://techcrunch.com/2011/07/17/hadoop-startups-where-open-source-meets-business-data/, 2011.

[24] D. BORTHAKUR, *The hadoop distributed file system: Architecture and design*, Hadoop Project Website, 11 (2007), pp. 21.

[25] C. CHU, S. K. KIM, Y. A. LIN, Y. Y. YU, G. BRADSKI, A. Y. NG and K. OLUKOTUN, *Map-reduce for machine learning on multicore*, Advances in neural information processing systems, 19 (2007), pp. 281.

[26] J. DEAN and S. GHEMAWAT, *MapReduce: a flexible data processing tool*, Communications of the ACM, 53 (2010), pp. 72-77.

[27] J. DEAN and S. GHEMAWAT, *MapReduce: simplified data processing on large clusters*, Communications of the ACM, 51 (2008), pp. 107-113.

[28] J. DEAN and S. GHEMAWAT, MapReduce: Simplified Data Processing on Large Clusters 6th Symposium on Operating System Design and Implementation, San Francisco (2004).

[29] R. LÄMMEL, *Google's MapReduce programming model—Revisited*, Science of Computer Programming, 70 (2008), pp. 1-30.

[30] C. RANGER, R. RAGHURAMAN, A. PENMETSA, G. BRADSKI and C. KOZYRAKIS, Evaluating mapreduce for multi-core and multiprocessor systems, High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on, IEEE, 2007, pp. 13-24.

[31] K. SHVACHKO, H. KUANG, S. RADIA and R. CHANSLER, The hadoop distributed file system, Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on, IEEE, 2010, pp. 1-10.

[32] T. WHITE, *Hadoop: The definitive guide*, O'Reilly Media, 2012.

[33] Vasudevan, M., & Deo, N. (2012). Efficient community identification in complex networks. *Social Network Analysis and Mining*, *2*(4), 345–359 LA – English. doi:10.1007/s13278-012-0077-5