A case study: Undergraduate self-learning in HPC including OpenMP, MPI, OpenCL, and FPGAs

Peter Jamieson
Department of ECE
Miami University
Email: jamiespa@miamioh.edu

Martin Herbordt and Michel Kinsy
Department of ECE
Boston University
Email: herbordt@ mkinsy@ bu.edu

***Abstract*—High Performance Computing (HPC) continues to develop and encroach on higher-education in the computing fields. With the ubiquitous availability and growth in commercial cloud computing and the diminishing performance returns on sequential programs, many developers must be able to understand and exploit parallel computing paradigms for certain applications. Focusing on Computer Engineering undergraduates, who arguably, will be future leaders in these parallel domains, the CE2016 recommended curriculum [1] has a number of hours dedicated for parallel and distributed computing where approximately 10 core hours are to be taught in parallel programming, other ideas are taught in and among networking and embedded systems, and an entire section on digital design (50 hours). In reality, this is not enough time to become competent in the broader HPC field, nor do we expect standard undergraduate curriculum to develop competent undergraduate into parallel programmers.**

However, as demand increases for HPC developers, one wonders how students will attain this knowledge. Many people learn HPC competencies in graduate work and industrial work, but what might be done early. In this paper, we look at what a developer can possibly learn in the HPC world, and what tools and understanding is needed to build and experiment with parallel implementations. Our goal is to look at aspects of HPC given the constraints of a typical laptop, and we ask what can a developer test and learn about on their system in the HPC domain. The benefits of this work is a better understanding of what tool sets students will need to understand to develop simple parallel implementations, what HPC platforms can be used for courses or personalized learning, and we provide a basic framework and code samples for people to start from.

I. Introduction

High Performance Computing (HPC) is a broad field that can include many computation architectures (including clusters, parallel machines, DSP processors, ASICs, and FPGAs) and many development platforms, programming models, and design languages [2]. Accelerating computation is complex, and for any given application and a target architecture it can take experts many weeks and months to improve performance (where performance can be a measure of execution time, power, cost, and other parameters).

As the need for accelerated computation increases, there is an accompanying need for developers who can work in this domain. So far, this need has been satisfied by the computing fields via training in graduate school and industrial settings. The question remains, can undergraduates prepare themselves to work in this domain, and if so, what tools and platforms can they, currently, learn about and access.

This work takes a look at this problem by taking a particular application, Conway's Game of Life [3], and implementing it across a number of typical (and less typical) HPC target platforms. This includes creating and running implementations of the benchmark in C on a single CPU, a shared memory C implementation using OpenMP on a number of CPUs, a message passing C implementation using MPI on a number of CPUs, a C OpenCL implementation targeting a CPU host and GPU execution engine, and a Verilog HDL (Hardware Description Language) implementation targeting an FPGA.

This work provides two additional artifacts. First, while doing this exercise we create a list of open source and commercial tools that were used, and that a learner should, probably, become competent with. Second, the result of this process is a sample benchmark with multiple implementations that a learner can access to model their own experimentation. Additionally, we provide some details on what type of benchmarking can be done within this HPC exploration for learning, but do not deal with the topic of optimization for these platforms. This provides a methodology to do basic benchmarking and comparison. This type of optimization work requires significant effort and is left to researchers and educators in each of these particular HPC target domains.

Finally, we provide some insights on working in this domain in terms of debugging and building benchmarks. This "wisdom" was gained by experiencing the exercise, and these lessons might save developers and learners some of their own time. We have extracted these insights from our work, but suggest to the learners that some insights are only learned by performing a similar exploration exercise. In this vein, we provide a suggested list of other applications to try to accelerate on the various platforms.

The remainder of this paper is organized as follow: Section II describes some background into what HPC and acceleration are and existing resources and research in this domain. Section III describes our benchmark and it's base implementation in C. Section IV we describe each of the accelerated implementations in terms of the target system and tools needed to implement them. Section V shows the benchmarking methodology we adopted and what results we could obtain for each of our designs. Section VI gives some context to this work,

and finally, section VII provides a conclusion to this work and future work.

II. Background - HPC and acceleration in Computer Engineering Education

HPC is a broad field that continues to expand as the demand for more and faster computation increases. The vast array of target devices and ways of designing for these is itself broad and there are a number of researchers who have examined performance differences among different platforms [4] [5]. In higher education, there have been a number of educators/researchers who have created, examined, and contributed to how to teach HPC to computer engineering undergraduates, which we hope to compliment.

In 1995, Nevison wondered how to include parallel computation in undergraduate education [6]. Since then, traditional parallel development on parallel systems has become more and more accessible to undergraduates because the price of parallel systems, including single chip multi-core systems [7], are more affordable for the average consumer. Even before this, researchers were proposing student educational experiences in MPI [8] and shared memory programming models [9]. Since then, parallel programming has been an educational research question for many including the following paper [10] (note that this is only a small set of researchers exploring this domain). The need to learn about parallel computing has gone beyond just computing majors and is a topic of interest for scientists and their simulations and models [11].

In addition to traditional parallel systems, the GPU has emerged as a popular acceleration device, and therefore, how to teach students to use these devices is of interest [12]. Similarly, FPGAs are programmable hardware fabrics that continue to compete with both CPU and GPU as a choice for application acceleration, and educators have wondered what should be taught on this target device [13]. Traditional VLSI knowledge has been pushed out of the computer engineering undergraduate curriculum and replaced with the need to teach FPGA and digital design skills with a focus on hardware description languages (HDL) [14], [15].

Finally, the emergence of cloud computing, which also has been contemplated in terms of education [16], researchers have began to question how a heterogeneous mix of devices can be used. For example, the OpenCL [17] programming model has been introduced where a host spawns off kernels on an array of heterogeneous cores including CPUs, GPUs, and FPGAs. This brings even more questions of what skills and experiences should be exposed to undergraduates.

Our work in this paper, as compared to more modern research differs in the main goal. If we look at recent parallel education research at major universities [18] (MIT) and [19] (ETH), then we see how these places and their curriculum can have a significant parallel education component due to human and computing resources, allowing them to ask what and how parallel concepts can be taught. Smaller universities [20] can still participate in this type of research, but, typically, the focus is on creating new tools and lessons that access HPC as resources are limited. Our work focuses on what low-cost approaches to working with HPC are possible at present, which we think will help the smaller schools while still allowing students at larger schools to work on their own parallel programming skills.

The HPC world continues to expand, and this means that the the skill set and knowledge that developers need is also expanding. Our work looks at what platforms are easily accessible for undergraduates to begin to explore and understand the development platforms and target devices independent of what their universities offer. This differs, significantly, from the above related work in that we are focusing on taking a snapshot of current technology and asking, "What can an undergraduate implement locally on their machine?". Previous work is focused on what can be taught in the university setting with the resources of the university, and what are the best methods to teach parallel concepts?

III. Conway's Game of Life as the application

Conway's Game of Life [3] is the application we implement on a number of parallel and acceleration platforms. This application is an example of cellular automaton that can also be classified as one of the simplest form of agent based simulation where each cell is it's own entity that is either ALIVE or DEAD at each time step of simulation based on some rules. The cells in our implementation are part of a 2D grid where each cell has eight neighbours.

Cell x is ALIVE or DEAD for time step $t+1$ based on the the following rules:

1) Any cell(x) that is ALIVE at t becomes DEAD at $t+1$ if less than two neighbours are ALIVE
2) Any cell(x) that is ALIVE at t becomes DEAD at $t+1$ if less greater than three neighbours are ALIVE
3) Any cell(x) that is ALIVE at t stays ALIVE at $t+1$ if their are two or three neighbours ALIVE
4) Any cell(x) that is DEAD at t becomes ALIVE at $t+1$ if exactly three neighbours are ALIVE

For our implementations of Conway's game of life, we consider the 2D grid to be a toroidal array such that the top and bottom rows are adjacent and, similarly, the left and right most columns are adjacent when calculating neighbour state. In terms of a computation problem, it can be considered a sliding window calculations [5] and is generally classified as a structured grid in the parallel dwarfs [21].

Our initial implementation is written in C and executed on a single CPU, and the program sequentially iterates through each cell and updates the respective state for each simulation tick or frame.

There are a few properties of importance in this application and we will describe them based on taxonomies from simulation [22] and agent based simulation [23]. Specifically, Conway's game of life is a closed, deterministic, homogeneous, discrete-event, time-driven simulation environment where the individual agent perceives locally, reasons/reacts partially, and communicates locally. The importance of this is how it impacts parallel implementations, and the key properties are the data

communication pattern is local, there is no dependency on the current time step (only the last time step matters), and the result of each time step needs to be recorded.

Table I shows a list of tools that were used in our experimentation. Undergraduates and learners who are attempting to become more proficient with parallel implementations should be familiar with many of the basic NIX and Windows command line tools as well as how to write, build, execute, and debug NIX and Windows programs written in C. Even this base of knowledge with tools, independent of parallel applications, is a significant body to learn. However, we believe this knowledge is a minimum requirement to work in traditional computing, and more and more of this knowledge should be pushed as early as possible in undergraduate education [24] for computer engineers.

IV. Parallel Implementations

We attempted to implement our design for 5 different acceleration platforms/systems including OpenMP for multiple CPUs, MPI for multiple CPUs, OpenCL targeting a CPU host and GPU target device, OpenCL targeting a CPU host and FPGA target device, and Verilog for an FPGA target device. All but one of these platforms can be worked on with a base laptop, which we will describe as our starting constraint.

The laptop used for these experiments, is a Dell Latitude E6510 from 2010 with a dual core i5 Intel Processor (2.7GHz) with 4GB of RAM. The GPU is an NVIDIA NVS 3100M with 512MB of memory. The operating system is Windows 10. On this machine, we have created a virtual machine (with Oracle VirtualBox) to run Linux machine (Ubuntu) with 4 processors and 1GB of RAM. These two systems provide the tools and capabilities to implement 4 of the 5 platforms listed above, and this device is typical of what undergraduates would have, noting that many students choose Apple products, which we have not explored (though virtual machines should not mean this is a barrier).

Table II shows each of the targeted devices. For each device, we name the implementation for future usage, describe the device, languages, machine requirements, challenge of using, and what type of execution of the system we performed. The OpenCL targeting a CPU host and FPGA target was the one platform that was not usable on our platform since the requirements were beyond our constrained laptop.

A. OpenMP targeting multiple CPUs

The first targeted accelerating device is OpenMP in Linux [25]. With the four cores available on the virtual Linux machine the code was written so that the work is shared across four threads. The grid is duplicated in shared memory for time step t and $t+1$, and there are no dependencies to calculate the cells state for time step $t+1$ as the neighbour data is all from time step t.

OpenMP is compiled with gnu gcc tools by linking the OpenMP library and using defined pragmas for spawning threads and implementing barriers around the simulation of each time step t so that the data can be written to the output file once all of the threads has completed calculations for their respective cells.

This parallel programming model was easy to get working for the Linux system, and the original code was modified with a few changes. The idea of shared memory is not too difficult to understand for developers, and the biggest challenge for learners in this domain is the idea of synchronization primitives for memory dependencies. However, the only memory dependencies for Conway's game of life is from time step to time step, which only requires barriers. As is the case in all of our implementations, debugging parallel applications can be challenging, but for this application very little debugging was needed and print statements were sufficient.

B. MPI targeting multiple CPUs

MPI was implemented and tested on the Linux system using MPICH-1 [26]. We, also, tested our implementation using MPICH-3, which is not a as simple to setup on a Linux machine. In general, an MPI application uses specific calls withing the MPI API to initialize the environment, and perform communication and synchronization actions. To build the system a specific compiler is used and the compiled code is initialized for N processes by a program in the MPICH environment.

For our MPI implementation, there are 4 processes (can be more or less) where the process with rank 0 initializes the grid and is responsible for writing the output of each time step to an output file. After reading the initial grid, the rank 0 process messages each of the other processes with their needed local data, and we partition this based on allocating contiguous rows to each process. At each time step, a process sends its' rightmost and leftmost rows to the proper process since they will need this data to calculate some of their neighbour information. Next, each process calculates the results for the time step and sends their updated cell state to the rank 0 process. Barriers are used to synchronize time steps and writing the output data.

MPI provides a low level parallel programming model where each process needs to explicitly send and receive data. The ordering of these is fundamental, and some of the ideas of "blocking" and "non_blocking" communication can be challenging to understand for learners. Getting the MPI system running on Linux is relatively easy, but debugging the MPI communication took longer than expected for our simple application.

C. OpenCL with CPU host targeting GPU

OpenCL is designed to allow heterogeneous acceleration systems to be created where there might be a mix of CPUs, GPUs, FPGAs, and other hardware target devices [17]. A number of steps are required to install the software development kit and drivers to get the OpenCL framework ready to compile, and we used Intel documentation, which was very explicit and useful in the setup. To create an application withing the OpenCL framework, a number of steps must be taken to setup a context of the platform and communication between the host

TABLE I
TOOLS USED IN OUR EXPLORATION

Tool	Use in this Work	OS	Cost
vim	text editor and code editor	NIX	Free
gcc	C compiler	NIX	Free
gdb	debugger	NIX	Free
valgrind	memory debugging	NIX	Free
CMake	create make files	NIX	Free
Make	make project	NIX	Free
Linux command line tools	basic NIX tools such as: ls, cd, etc.	NIX	Free
Window command line tools	basic Windows tools such as: dir, cd, etc.	Windows	Purchase
Visual Studio 15	build windows software	Windows	Purchase
Intel OpenCL sdk	compile OpenCL for GPU	Windows	Free
Intel OpenCL Offline Compiler	compile OpenCL kernel	Windows	Free
Processing	create simple graphic tools	Windows	Free
Github Desktop	github interfacing tool	Windows	Free
Intel's Quartus Prime Lite (17.0)	Building, Simulating FPGA designs	Windows	Free
Python	scripts and simple tools	NIX/Windows	Free
PDF reader	read pdf documentation	NIX/Windows	Free

TABLE II
DEVICES AND PLATFORMS USED AND REQUIREMENTS

Name	Device	Language	Machine Requirements	Challenge	Actual Execution
C-imp	CPU	C	Low	Low	Virtual Machine
OpenMP-imp	CPUs	C - OpenMP	Medium	Low	Virtual Machine
MPI-imp	MPI	C - MPICH-1	Medium	Medium	Virtual Machine
OpenCL-gpu	GPU host(CPU)	C/C++ - OpenCL	High	High	Real
Verilog-imp	Verilog FPGA	Verilog	Low	High	Simulated
OpenCL-fpga	FPGA host(CPU)	C/C++ - OpenCL	Very High	High	None

and heterogeneous devices. Additionally, the kernel code (code that will execute on the devices) needs to be compiled using a separate device specific compiler, which can make it difficult to debug the kernel code. There are a number of introductory books and example code is available online to help with this process.

For our implementation, we spawn individual kernel threads on a GPU for each cell. The host is responsible for synchronization of each time step and writes out all the data to an output file.

This is a very simple model of execution, but getting our simple application working in this framework took considerable time because of the complicated initialization steps, and then debugging the application. This was a much more difficult platform to work with compared to the previous systems, however, OpenCL is the newest of the systems explored in this work by at least 10 years. Therefore, in the future if this platform gains traction, we can expect much more support for building these types of systems.

D. Verilog targeting an FPGA

FPGA based acceleration is a different approach compared to all the previous C based fixed target systems with more traditional parallel programming models. The FPGA is a reprogrammable chip which can implement general purpose computation devices such as CPUs, but the hardware can be tailored exactly for the computation. This means that learners need to have a significant understanding of digital system design and how to design applications using Hardware Design Languages (HDLs) such as Verilog [27]. Not only does a user need to understand digital design, the platform/tools for developing, simulating, and programming an FPGA have some similarity to the programming tools, but have an additional learning curve. The two most common used platforms are from Xilinx and Intel, and in our case, we use the freely available free version of Intel's Quartus Prime Lite and the accompanying tutorials.

There are a number of choices to make when the underlying silicon substrate is a programmable logic array such as an FPGA. For our application, we chose to make each cell of the 2D grid a Verilog module that includes logic to be initialized, communicate state, and update state. To make the 2D array of these cells, we wrote python scripts to automatically generate the Verilog with direct connections between the cell and its' neighbours. To observe the state and control the array of cells, we wrote a control module (as a finite state machine) that initializes, simulates a step, and reads the state of all the cells. Finally, we wrote a script in Python to create testbenches to test the implementation by synthesizing our design for an Intel Cyclone V FPGA, which is a simple small FPGA, but is targetable by the Lite software package.

The complexity of this design is not great, but because of the design environment and knowledge needed to create designs for FPGAs in Verilog is significantly different from the other implementations, we consider this process to be "high" in

terms of complexity. Fortunately, digital design is, typically, part of a computer engineering undergraduate curriculum, but for other learners with less of these skills this type of exploration is considered the most challenging to learn in our exploration.

E. OpenCL with CPU host targeting FPGAs

The last target acceleration system targets FPGAs with OpenCL that might allow for developers more familiar with C to still accelerate their designs on an FPGA. This is the newest platform in terms of development, and at present the basic requirements to even emulate these systems is far beyond the capabilities of the laptop we were constrained by. Also, to run the system, very specific FPGA prototyping boards are needed. Currently, this type of system is not ready for undergraduates to learn and experiment with unless they have access to more advanced systems.

OpenCL targeting FPGAs makes it a High-Level Synthesis (HLS) language, and there exist a number of competing languages in this space with the greater goal of C-to-gates. There are a few languages and IDEs in this space that are freely available for students, but they typically are tied to specific development boards. Our attempts with OpenCL were limited by the capabilities of the laptop. Other languages are limited by what systems they target. This space is accessible by undergraduates, but we do not believe this is a easily targetable learning space for undergraduates on their own. This will, likely, change in the coming years.

V. Basic Results

In Table II, we can see from the final column that most of our implementations are either executed on a virtual machine or are a simulation. The exception is the OpenCL-gpu design that is run on the laptop's hardware. From a benchmarking perspective, we can not make direct comparison of performance between each implementation, and we should note that we don't attempt to optimize each implementation for its' target platform as this takes significant time, which is better described in Xilinx's technical report on creating FPGA designs [28].

The methodology we use to benchmark our designs focuses first on determining for each of the systems what is the largest possible implementation of our benchmark on the target in terms of the dimensions of the 2D square grid by testing in increments of 2500 cells in both the X and Y direction. For all the software approaches this works where the designs can fit a maximum size of either 7500by7500 or 10000by10000. We report the FPGA implementation in the text since it is significantly smaller. In terms of these benchmarks, we vary the grid size, and we create a benchmark by inserting gliders (which are oscillating Conway entities) into the benchmark every 6 by 6 tile within the 2D grid.

Table III shows the results for each of the software implementations. As stated earlier, the first three rows are for the benchmark executed on a virtual Ubuntu CPU in C, OpenMP, and MPI with times measured using clock, omp_Wtime, and mpi_Wtime, respectively. The OpenCL-gpu results are measured directly on the hardware using ftime to measure time. All times are in milliseconds, and two times are reported where the first is the total execution time (without initialization), and the second is the time spent writing the current state to file.

As we discussed, we're not trying to compare the different implementations and optimize them, but we see that for our implementations very little speedup (if any) is gained from the parallel versions as compared to the single core C implementation. From a student perspective, this may illustrate how naive parallel implementations don't, necessarily, provide a benefit to the application, and a deeper understanding of how to structure designs for different parallel paradigms takes significant effort [28].

The FPGA implementation in Verilog is purely a simulation. We fit a 100by100 instance of the benchmark on a Cyclone V (5CSXFC6D6F31C6) where the logic utilization (in ALMs) is 92% of the chip(38,467 ALMs of 41,910 ALMs) with 23819 registers. This design, however, does not capture full I/O output to file as the above one and simply outputs the design via a port on the FPGA. Therefore, this system is not complete in terms of an application with host and co-processor capabilities.

VI. Lessons Learned and Application to HPC Education

In this case study, we note that the experimenter's experience is in Verilog design with brief work on OpenMP in the early 2000s, and the experience is much more significant than the average undergraduate. The key lesson, however, is that students can access parallel programming with only their computation tools, and there is no need to work on all the systems described in this paper.

A learning computer engineer should pick a parallelizable application and implement it on at least one parallel model and system as described above. We would argue the shared memory model is the easiest starting point with either OpenMP or another thread library. Then profiling the application and attempting to optimize their implementation will give any learner a significant experience with parallel development. They can then either attempt to implement the system on a different parallel model or try another application that has different characteristics. By accumulating these experiences the learner will soon become more and more competent with parallel thinking and design.

We have no evidence that shows these experiences are the best way to learn about HPC development, but in our experiences we have not met skilled parallel developers, in industry or academia, who haven't learned their programming skills without practice. This paper shows that students can access and practice in most leading edge parallel systems, with the exception of heterogeneous parallel systems and HLS languages and tools.

TABLE III
TIMES FOR DIFFERENT BENCHMARKS

Name	2500by2500	5000by5000	7500by7500	10000by10000
C-imp	880, 290	3225, 1307	7360, 3408	16494, 8716
OpenMP-imp	849, 317	4252, 1413	11478, 4728	20381, 6622
MPI-imp	741, 466	2576, 1734	23736, 8931	NA
OpenCL-gpu	2960, 399	10051, 1612	23177, 3154	NA

VII. CONCLUSION

Learning HPC design implementation on CPUs, GPUs, and FPGAs is possible on a traditional laptop. We demonstrated this with our one off exploration of Conway's Game of Life implemented on a number of platforms. This means that students with enough NIX skills can, likely, build simple naive applications for these devices to start their learning process, and courses in this domain do not need specialized equipment to teach the process. The latest and greatest innovations in HPC that include heterogeneous mixtures of computational chips, however, is not accessible to students yet. This is not surprising as these heterogeneous systems are still on the cutting edge of technology as of the writing of this paper. The next step in this work is to understand how to expose and teach students to optimize their applications.

For a learning student, we suggest trying a similar exercise as above focusing on a simple algorithm from one of the 13 dwarfs [29] and implementing it on a number of different architectures. We suggest choosing an N-body problem such as the million-star simulation.

REFERENCES

[1] J. Impagliazzo and *et.al*, "Curriculum guidelines for undergraduate degree programs in computer engineering," 2016. [Online]. Available: https://www.acm.org/binaries/content/assets/education/ce2016-final-report.pdf

[2] P. Jamieson, A. Sanaullah, and M. Herbordt, "Benchmarking heterogeneous hpc systems including reconfigurable fabrics: Community aspirations for ideal comparisons," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 2018, pp. 1–6.

[3] M. Gardner, "Mathematical games: The fantastic combinations of john conways new solitaire game life," *Scientific American*, vol. 223, no. 4, pp. 120–123, 1970.

[4] L. Yang, S. C. Chiu, W.-K. Liao, and M. A. Thomas, "High performance data clustering: a comparative analysis of performance for gpu, rasc, mpi, and openmp implementations," *The Journal of supercomputing*, vol. 70, no. 1, pp. 284–300, 2014.

[5] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*. ACM, 2012, pp. 47–56.

[6] C. H. Nevison, "Parallel computing in the undergraduate curriculum," *Computer*, vol. 28, no. 12, pp. 51–56, 1995.

[7] R. Brown, E. Shoop, J. Adams, C. Clifton, M. Gardner, M. Haupt, and P. Hinsbeeck, "Strategies for preparing computer science students for the multicore world," in *Proceedings of the 2010 ITiCSE working group reports*. ACM, 2010, pp. 97–115.

[8] E. M. Minty and M. Westhead, "Mpi on-line: A teaching environment for mpi," in *Supercomputing'97*, 1997.

[9] Y. Pan, "Teaching parallel programming using both high-level and low-level languages," *Computational ScienceICCS 2002*, pp. 888–897, 2002.

[10] A. Breuer and M. Bader, "Teaching parallel programming models on a shallow-water code," in *Parallel and Distributed Computing (ISPDC), 2012 11th International Symposium on*. IEEE, 2012, pp. 301–308.

[11] D. A. Joiner, P. Gray, T. Murphy, and C. Peck, "Teaching parallel computing to science faculty: best practices and common pitfalls," in *Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 2006, pp. 239–246.

[12] C. El Amrani, "A learning approach to introducing gpu computing in undergraduate engineering program," *International Journal of Computer Applications*, vol. 107, no. 20, 2014.

[13] P. Schaumont, "A senior-level course in hardware–software codesign," *IEEE Transactions on Education*, vol. 51, no. 3, pp. 306–311, 2008.

[14] S. Areibi, "A first course in digital design using vhdl and programmable logic," in *Frontiers in Education Conference, 2001. 31st Annual*, vol. 1, 2001, pp. TIC –19–23. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.3048

[15] P. Jamieson, L. Grace, B. Zhang, and N. Mizuno, "verilogtown-improving students learning hardware description language design-verilog-with a video game," in *2015 ASEE Annual Conference and Exposition*, 2017.

[16] N. Sultan, "Cloud computing for education: A new dawn?" *International Journal of Information Management*, vol. 30, no. 2, pp. 109–116, 2010.

[17] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in science & engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[18] C. Ivica, J. T. Riley, and C. Shubert, "Starhpcteaching parallel programming within elastic compute cloud," in *Information Technology Interfaces, 2009. ITI'09. Proceedings of the ITI 2009 31st International Conference on*. IEEE, 2009, pp. 353–356.

[19] T. R. Gross, "Breadth in depth: a 1st year introduction to parallel programming," in *Proceedings of the 42nd ACM technical symposium on Computer science education*. ACM, 2011, pp. 435–440.

[20] B. Rague, "Teaching parallel thinking to the next generation of programmers," *Journal of Education, Informatics and Cybernetics*, vol. 1, no. 1, pp. 43–48, 2009.

[21] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams *et al.*, "The landscape of parallel computing research: A view from berkeley," Tech. Rep., 2006.

[22] A. Sulistio, C. S. Yeo, and R. Buyya, "A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools," *Software: Practice and Experience*, vol. 34, no. 7, pp. 653–673, 2004.

[23] L. J. Moya and A. Tolk, "Towards a taxonomy of agents and multi-agent systems," in *Proceedings of the 2007 spring simulation multiconference-Volume 2*. Society for Computer Simulation International, 2007, pp. 11–18.

[24] P. Jamieson and J. Herdtner, "More missing the boat - arduino, raspberry pi, and small prototyping boards and engineering education needs them," in *Frontiers in Education Conference (FIE), 2015. 32614 2015. IEEE*. IEEE, 2015, pp. 1–6.

[25] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming," *IEEE computational science and engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[26] E. P. Computing, "A high-performance, portable implementation of the mpi message passing interface standard," *Parallel Computing*, vol. 22, pp. 789–828, 1996.

[27] *Verilog Hardware Description Reference*, Open Verilog International, March 1993.

[28] Xilinx, "Introduction to FPGA Design with Vivado High-Level Synthesis," Xilinx Inc, Tech. Rep., 2013.

[29] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. D. Kubiatowicz, E. A. Lee, N. Morgan, G. Necula, D. A. Patterson *et al.*, "The parallel computing laboratory at uc berkeley: A research agenda based on the berkeley view," 2008.