Puzzle Solver Accelerators Make Excellent Capstone Design Projects

Soheil Ghiasi, Matin Hashemi, Volodymyr Khibin, Faisal Khan
Department of Electrical and Computer Engineering
University of California, Davis
CA, USA 95616
{ghiasi, hashemi, vykhibin, fnkhan}@ucdavis.edu

***Abstract*—We present a computer engineering capstone design project course focused on accelerating intensive computations via integration of application-specific co-processors with digital processor systems. We propose utilization of puzzle solvers as attractive, scalable and simple-to-understand applications to engage students with practicing a number of fundamental concepts in algorithm design, HW-SW co-design, computer architecture, and beyond. While advocating a contest setup for the course, we discuss several well-specified milestones that enable balancing students' creativity and freedom in design choices with ensuring timely progress towards the end goal of the class. We report our observations with the only offering of the class so far, which resulted in successful project completion by all students, and their supportive feedback.**

Keywords-Capstone Project, Computation Acceleration, Puzzle Solvers, HW-SW Co-Design, Solution Space Exploration

I. INTRODUCTION

The capstone project has become an important part of the engineering education in the US. In particular, ABET [1] guidelines require the capstone project to engage teams of senior students in multi-disciplinary projects of sufficient complexity. One of the objectives is to give students an opportunity to practice a range of material and techniques that they have been taught throughout the undergraduate program.

Development of a quality capstone course is challenging due to a number of reasons. The major challenge arises from the natural tradeoff between complexity and manageability in academic settings. On one hand, the project has to be substantial enough to expose students to a number of fundamental concepts, and to provide them with an opportunity to practice such concepts. On the other hand, the scope, resource requirement and breadth of underlying disciplines have to be reasonable to admit solutions by small teams of senior students under limited time in academic settings. In addition, an ideal project would have tangible specification and quality metrics, and would be appealing to many students to attract their attention.

We present a capstone design project that we have recently developed at the department of electrical and computer engineering at the University of California, Davis. The course, EEC181A/B, is offered over two consecutive quarters. It mainly targets students whose primary area is computer engineering, although many students with computer science or electrical engineering emphasis also participate in the course.

At its core, the project tasks teams of students with FPGA-based development of application-specific processors for acceleration of well-defined intensive computations. During the course of the project, students are exposed to a number of topics from their previous courses such as algorithm design and complexity analysis, HW-SW Co-design, computer architecture, parallel processing, quantitative benchmark-driven design decisions, and digital system development and validation.

We believe systems for solving logical puzzles, such as the well-known Sudoku or Hitori (discussed in this paper), make excellent project topics for a number of reasons. First, puzzles are easy to explain and navigate, yet they expose students to a wealth of relevant topics from algorithms and solution space exploration to parallelization and acceleration aspects in a digital system. Moreover, many puzzles can be arbitrarily scaled to showcase the role of algorithm complexity and memory bottlenecks in practical settings. Finally, our experience is that many students find puzzles attractive on a personal level, and relate to the problem domain (e.g., candidate solution and search space), tradeoffs (e.g., algorithms for solution space exploration), design decisions (e.g., computation kernels for HW acceleration) and quality metrics (solver runtime) very well.

Our contention (and subsequent observation) is that contests bring out the best in most students, by encouraging them to come up with creative ideas and to work harder to excel in the competition. As such, we have set up the course as a contest by providing the same problem specification and resources to all student teams, and by defining an open-ended metric (speedup over a minimum expected performance) to evaluate the quality of projects.

Our experience and the students' comments with the only offering of the class are very positive. We would be happy to share the teaching resources with interested colleagues.

II. THE HITORI PUZZLE

Hitori is an entertaining logic puzzle that was originally introduced by the Japanese game publisher Nikoli [2]. It is played with a grid of cells in which, each cell contains an integer number. The objective of the game is to eliminate numbers by filling in the cells (marking them as "black") such that the remaining cells (whites) do not contain numbers that appear more than once in any of the rows or columns.

In addition, filled-in cells cannot be horizontally or vertically adjacent, although they can be diagonally adjacent.

978-1-61284-639-2/11/$26.00 ©2011 IEEE

The remaining un-filled cells must form a single geometric component that is connected horizontally or vertically. An example 5x5 Hitori puzzle and a possible solution are shown in Figure 1.

The Hitori grid always has equal number of rows and columns. The number of rows, or columns, of the grid defines the *order* of the puzzle through which, the problem can be arbitrarily scaled. The numbers in the cells are always positive integers no larger than the order of the puzzle. That is, the numbers used in the cells are always between 1 and the order of the puzzle. The example in Figure I illustrates an order-5 Hitori puzzle, a correct and an invalid solution (not a single connected component).

We ask students to develop an FPGA-based Hitori solver that can solve puzzles of orders 5, 8, 12, 20, 50 and 100. The puzzle orders are chosen to cover a wide range of complexities, which might call for somewhat different tweaks to the solver algorithm and its implementation.

III. HW PLATFORM

All teams are given an Altera DE2 FPGA board and copies of the design software, which were made possible by a generous donation from Altera [3]. The DE2 board is the only platform that can be used for development of the solver system. Identical hardware resources and problem specification provide a level playing field for all participating teams in the competition.

IV. MILESTONE I: PUZZLE SOLVING ALGORITHM

To ensure timely progress of student teams, we loosely identify a set of milestones that teams have to demonstrate by certain dates. The first milestone involves development of the solver algorithm and its software prototype, for which students are free to use any technique of their choice.

An interesting observation was that although teams started with rather different approaches to solver algorithms, they arrived at rather similar (from a high-level approach viewpoint) general conclusions, after researching, prototyping and experimenting with different puzzle testbenches. The common approach, which is quite intuitive, is to deterministically reduce the solution space (e.g. via logical deductions) and only resort to exploration if needed.

A. Deterministic Logical Deduction

One can logically infer a number of easy-to-implement feasibility checks or deduction steps to deterministically reduce the solution space. For example, if a particular cell is already marked black (as a result of another deduction algorithm or guess), all of its adjacent cells must be marked white, as black cells cannot touch in any valid solution.

As another example, if three identical numbers are adjacent in the same row (or column), then the two outer cells have to be marked black to eliminate repetitions of the middle number in that row (or column) without marking two adjacent cells black.

Such logical inference rules are straight forward to implement in software, and reduce the complexity of the solution space substantially more efficiently than back tracking based exploration of the solution space.

B. Orderly Exploration of the Solution Space

There tends to be a limit on the applicability of the checks for difficult puzzles, although ideally, one would like to deterministically arrive at a valid solution only by going through a series of logical deduction steps. This is not always possible either because the input puzzle is not designed to be solved only by deduction, or because complicated (and rarely applicable) logical inference routines are better skipped in favor of efficient exploration of the solution space via guessing.

In either case, a partial solution obtained via logical deduction has to be further solved by orderly exploration of the solution space. At this stage, teams have to understand some important concepts in search of the solution space, such as solution space complexity, backtracking and exploration tradeoffs (e.g. ordering of guesses). Students also learn that the two guessing and logical deduction methods can be combined to quickly check the feasibility of a guess and to avoid unnecessary subsequent guesses.

V. MILESTONE II: DESIGN PLANNING

Following development of a software prototype, students were tasked with development of a design plan for accelerating the solver's algorithm via hardware-enabled parallelization and HW-SW co-design of the system. The design plans are presented both in writing and orally to enable the instructors provide necessary feedback early in the design process.

A. Profiling and Kernel Identification

Stressing Amdahl's law, students were asked to perform timing profiling of their software prototypes to identify compute-intensive kernels of their algorithms. Interestingly, the timing profiling and analysis helped the teams to optimize their software implementations to arrive at faster runtimes. The polished software prototypes are used as the baseline software implementation for each team.

Most teams found out that simple logical inference routines form the kernels of the algorithm, as they should be applied iteratively to deterministically reduce the search space. Note that after sufficient application of a primary inference routine, a secondary logical inference or even a guess might enable the application of the primary inference again, although it could no longer advance the solving process on its own.

B. Accelerator Design Blueprint

Subsequently, each team developed a blue print of the accelerator hardware and its interface with software. The hardware blueprint has to include RTL-like details, and hence, should cover information on anticipated data encoding, HW-SW interface, and bus access arbitration between custom HW module and the processor.

C. Performance Estimation

The last component of the design plan was performance estimation of the solver system based on the system blueprint and software timing profile. Specifically, teams were asked to estimate the latency of the HW-mapped kernel with semi cycle accuracy (cycle accurate for intensive loops), and estimate the performance of the system if it were to be implemented. Any team whose design plan did not meet the performance target (see Section VII) was sent back to the drawing board to improve their design plan.

Our experience is that the design plan is likely to be the perfect stage for the instructor to step in, and provide feedback to steer students away from wasting excessive amount of time and energy on a design that is unlikely to meet the specification.

VI. Milestone III: HW+SW Co-Implementation

Following review, discussion and approval of the design plans, teams proceed with implementation and debugging of their proposed designs. The implementation process is typically the most time consuming milestone, as students have to debug many subtle issues that come up with interfacing HW and SW, access to shared resources, and synchronization of concurrent HW and SW modules.

As part of this stage students get to practice many engineering techniques used for debugging. In addition to conventional simulation, and divide & conquer-based debugging approaches, teams found Altera SignalTap particularly helpful, as it enables monitoring of design signals after they are mapped to the FPGA.

VII. Project Evaluation Methodology

We specify a time budget for solving a testbench puzzle of a particular order, which specifies the expected runtime of the solver for finding the solution. The time budget for solving a puzzle of order N is $60*N^3$ µ-seconds, which is arrived at by a combination of solver complexity analysis, empirical adjustments, and expected HW speedup over software prototypes. Students found the expected speedup (time budget) to be about an order of magnitude relative to their polished software implementation (Milestone I). The formula limits the expected runtime for $N=5$ to be 7.5 milliseconds, and for $N=100$ to be 1 minute.

The designs are partially judged based on the time it takes to solve puzzles. In particular, the designs receive a Performance Score (PS), which is essentially the geometric mean of speedups over all puzzles relative to their expected time (time budget):

$$PS = \left(\prod_{all\_puzzles} \frac{Expected\_Runtime(N)}{t_N} \right)^{\frac{1}{no\_of\_puzzles}}$$

, where N refers to the puzzle order, *Expected_Runtime(N)* is the expected time for solving puzzles of order N, and t_N is the time the solver needs to solve a given puzzle of order N. Both time budget and solver runtime are measured in tenths of millisecond. Note that a performance score of 1 indicates that on average, the solver *clears the bar* set for the class.

The rational for using the geometric mean of the speed ups is to equally weigh the speedup for different puzzles, rather than an arithmetic mean which would be biased towards larger numbers [4]. Note that the slowdown of runtime relative to the expected runtime for a particular puzzle diminishes the contribution of the speedup over another puzzle to the overall performance score.

If a solver fails to solve a given puzzle within 5X of the expected time budget, the partial contribution of that puzzle to the performance score is assumed to be 1/10. That is, the particular unsolved puzzle will penalize the geometric average of the speedups by a factor of 10. The rationale is to give a very high weight for correctly solving all puzzles, rather than giving up on a particular order and making up for it by over-optimizing another puzzle order.

A. Puzzle Generation

Since scaled puzzles are solely meaningful for computerized solution (and hence they are publicly unavailable), we had to develop a mechanism for generation of puzzles of different orders and difficulty levels to sufficiently evaluate students' designs. Subsequently, we developed algorithms for puzzle generation, which were implemented in a software program by the teaching staff.

The generated testbench puzzles were divided into two sets: the debug set and the evaluation set. The debug set puzzles were released to class to assist students in development, debugging, and benchmarking of their designs. The evaluation set was not disclosed to the students, and was merely used to evaluate their final designs. The objective was to eliminate the possibility of over-optimizing solvers to better fit particular puzzle testbenches.

B. Evaluation Software

We developed a software module that runs on a host PC that is interfaced with the FPGA board via a standard RS-232 connection. The software communicates with the design with a well-documented protocol. It sends puzzles of different orders to the FPGA, receives the solved puzzles and checks the solution for correctness. Also, it measures the solver runtime for each puzzles by which, it calculates the performance score (PS) of each participating team.

VIII. Teams' Performance and Feedback

All seven participating teams in the course were successful in developing designs that correctly and efficiently solved the input puzzles of different difficulty levels within the expected runtime. We evaluated the performance of the designs with four *undisclosed* puzzles of each order (5, 8, 12, 20, 50, and 100). The four puzzles of the same order represented a range of difficulty levels.

Out of 168 tests that we ran (7 teams, 6 orders and 4 puzzles per order), only one test failed due to the solver generating an invalid solution. It turned out that a particular input triggered a synchronization bug in one of the designs, which had not appeared previously when students tested with

debug benchmarks and it also did not appear with the remaining 23 tests of the evaluation benchmarks. Similarly, we had to stop only one test because the solver did not return any solution within the 5X timeout that we had specified for solver runtime. All the remaining 166 tests returned correct solutions within the 5X timeout with the overwhelming majority running significantly faster than the expected time. In fact, some puzzles were solved about 30X faster than the expected runtime (50X in case of debug benchmarks).

Table I reports the summary of performance scores, the geometric mean of measured speedups relative to the expected runtime, as obtained with both testbench (disclosed to students and used for design development) and evaluation (undisclosed and solely used for evaluation upon submission of designs) puzzles. The results are sorted with respect to PS under evaluation puzzles.

The first observation is that all teams achieved the performance target of the class by successfully accelerating their baseline software prototype using application-specific accelerators. The average speedups were as high as 7X more than the expected runtime, which was set to translate to about 10X improvement over baseline software implementation. Moreover, with the exception of team 5, the performance score measured over a wide range of shared testbench puzzles followed the same trend as PS over evaluation puzzles.

Our contention is that the large number and diversity of testbench puzzles (and evaluation puzzles) did not allow easy utilization of puzzle-specific hacks and to a large degree eliminated the indeterminacy of runtime with respect to input puzzles. This in turn, translated to a more fair and predictable contest, which was appreciated by students. Before the final evaluation, many students were concerned that a specific choice of evaluation benchmarks might render their design decisions unjustified (as is the case in real life!). We were happy to see that the ordering of teams was mostly preserved, while the important point about benchmark-driven design decisions hit home with students.

Students' feedback was quite encouraging. Many students made positive comments about the wealth of topics that were involved in the course of project, its contest setup, use of puzzle solvers as an appealing design problem, and the end-to-end nature of the project, which enabled them to practice their knowledge.

TABLE I. PERFORMANCE SCORES OF THE PARTICIPATING TEAMS

Team	PS (testbench puzzles)	PS (evaluation puzzles)
1	7.54	7.31
2	7.17	6.11
3	5.73	4.94
4	4.77	4.44
5	6.45	4.40
6	4.55	3.89
7	2.43	2.37

IX. CONCLUSIONS

We presented a computer engineering capstone project course on HW-SW co-design that utilizes puzzle solvers as a computationally-intensive application driver. The course allows students to practice a large number of fundamental concepts that are covered in a typical computer engineering curriculum. We showcase the use of a series of milestones that balance students' freedom and creativity with ensuring their timely progress towards the end goal.

ACKNOWLEDGMENT

We would like to thank Altera for the generous donation of FPGA boards and design software, which enabled us to offer this course. Lance Halsted helped us with the development of some laboratory material that introduced students to the hardware platform at the beginning of the course.

REFERENCES

[1] Accreditation Board for Engineering and Technology (ABET), http://www.abet.org/

[2] Nicoli (The First Puzzle Magazine in Japan), http://www.nikoli.co.jp/en/

[3] Altera Corporation, http://www.altera.com

[4] D. Patterson and J. Hennessy, "Computer Organization and Design: The Hardware/Software Interface", Morgan Kaufmann, 200

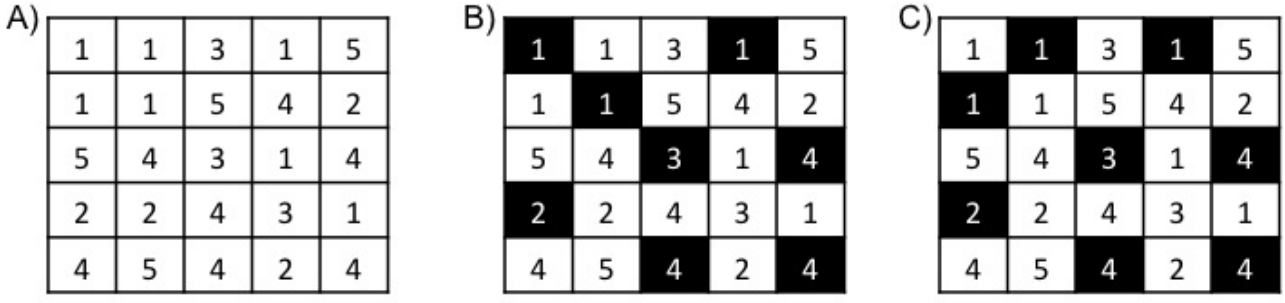

Figure 1. A)An example Hitori puzzle of order 5. B)A valid solution. C)An incorrect solution (not a single connected component)