AC 2012-3055: PARALLEL SIMULATION OF MANY-CORE PROCESSORS: INTEGRATION OF RESEARCH AND EDUCATION

Prof. Tali Moreshet, Swarthmore College

Tali Moreshet is an assistant professor at the Department of Engineering at Swarthmore College. Her research interests are in computer architecture, energy-efficient multiprocessor, many-core, and embedded systems. Her research is funded by NSF. Tali Moreshet earned a B.Sc in Computer Science from Technion, Israel Institute of Technology and a M.Sc. and Ph.D. in Computer Engineering from Brown University.

Prof. Uzi Vishkin, University of Maryland, College Park

Uzi Vishkin has been Professor of Electrical and Computer Engineering and permanent of the University of Maryland Institute for Advanced Computer Studies (UMIACS) since 1988. He was affiliated with Tel Aviv University between and 1984 and 1997, was Chair of CS there in 1987-8, and also worked for IBM T.J. Watson and New York University.

His research interests center around parallel algorithms and architectures. Facilitating a transition into ubiquitous parallel computing has been a strategic objective for computer science and engineering since its inception in the 1940s. A theory enthusiast, the overriding theme guiding his work was using theory to guide the rest of the field in addressing this strategic objective. Key components in his comprehensive plan include: (i) the very rich PRAM parallel algorithmic theory, and (ii) a PRAM-On-Chip vision comprising the explicit multi-threaded (XMT) computer system framework he invented. The latter provides a powerful approach to multi-core architectures, and, in particular, the exponential increase in the number of cores in the roadmap of most vendors into the late 2010s. Recently, he has focused on the feedback loop between algorithms, their programming and implementation, and architecture, as well as their performance modeling with an eye towards softer aspects, such as their ease-of-programming, teachability and learnability.

Dr. Fuat Keceli, Intel Corporation

Parallel Simulation of Many-core Processors: Integration of Research and Education

Abstract

Providing undergraduate students with an opportunity to experience meaningful academic research has a potential impact on their future career choice. Our approach combines two seemingly contradicting attributes: (i) to make it exciting, the effort targets a grand research objective; and (ii) to make the experience self-assuring and overall positive, the concrete task handed to a student is feasible, given their background and time constraints, while still contributing towards the grand objective. We believe that this can motivate a wider range of undergraduate students, including underrepresented groups of undergraduate engineering students to pursue an engineering career path, academic or otherwise.

In this paper, we describe a pilot of an on-going, multiple-year research project, carried out by undergraduate female students incorporating research and education in computer science and engineering (CS&E). Many-core processors are becoming increasingly popular in general-purpose computing. While most researchers agree that this requires introduction of parallelism to mainstream CS&E practice, and hence education, parallel programming difficulties remain obstacles that are yet to be overcome. For concreteness, the research project involves a certain many-core framework, called eXplicit Multi-Threading (XMT). The XMT framework provides a general-purpose many-core architecture for fine-grained parallel programs that scales to a thousand lightweight cores, aiming to improve single task execution time through parallelism. What makes XMT attractive is that it has been supported by significant evidence on ease-of-programming and competitive performance. The XMT platform consists of a proof-of-concept 64-core FPGA and ASIC prototypes and a highly configurable cycle-accurate simulator (XMTSim), capable of modeling a target 1024-core XMT.

Our work aims to parallelize XMTSim. (i) The grand objective is to establish that XMT is an effective self-simulating machine; namely, to efficiently simulate the XMTSim code by XMTSim itself. What makes this objective grand, and therefore inspiring, is that one of the elegant features of Turing machines was their ability to provide self-simulations. This feature has been used in support of the thesis that Turing machines are general-purpose. (ii) The combined milestone for the student projects is parallelizing the most computationally time-consuming component of XMTSim, the Interconnection Network (ICN) of XMT. (iii) Each student is being given a part of the job.

Introduction

Research opportunities for undergraduate students are becoming more common, and undergraduate students often expect to experience academic research before they graduate. Moreover, a positive and meaningful experience with engineering research contributes to the motivation of these students to pursue a future career in engineering and computer science, whether in graduate school or industry. Yet, finding ways of involving undergraduate students in active research introduces many challenges (e.g. [4]). Furthermore, ensuring that the research experience benefits the students as well as advances the research is an even greater challenge.

Our research involves parallel computing in the form of many-core processors. In order for an existing program's performance to improve with new generations of multiprocessors, the program needs to be parallelized. However, extracting parallelism from a serial task, and parallel programming in general, is a major challenge to the software industry and anyone using computers [5].

The undergraduate students that are involved in our research have limited time to dedicate to research, and usually come with little preparation. These factors add to the inherent complexities of parallel programing. The eXplicit Multi-Threading (XMT) general-purpose many-core platform for fine grained parallel programs makes an attractive framework given the above challenges. XMT is easy to teach, facilitating a smoother learning curve for students that are new to parallel programming and have limited background in parallel architectures [9].

In this work, we aim to parallelize the cycle-accurate simulator of XMT, XMTSim, with the following goals:

1. Establish that XMT is an effective self simulating machine; namely simulating efficiently the XMTSim code by XMTSim itself.
2. Parallelize the most computationally time-consuming component of the simulation, the Interconnection Network (ICN).
3. Carving out the parallelization task into small projects suitable for a single undergraduate student to pursue in one summer.

This paper presents a pilot study intended to reinforce the claim that XMT is easy to teach, by demonstrating that the students were able to learn to parallel program well enough to advance our research goal within the limited time scope. Our preliminary results already show that parallelizing the ICN obtains simulation speedups of x54 compared to the best serial implementation on a 64-core XMT. A large factor that enabled us to achieve this goal was breaking down the project into smaller, more manageable components.

Computer engineering and science is underrepresented by certain groups, mainly women and minorities. There is an increasing effort to attract these groups, with one example being DREU: distributed research experience for undergraduates from underrepresented groups [3]. We believe that XMT makes for a good medium to attract these groups, or to retain them in the field.

The XMT Many-core Platform

The primary goal of the eXplicit Multi-Threading (XMT) general-purpose computer architecture [10] has been improving single-task performance through parallelism. XMT was designed from the ground up to capitalize on the huge on-chip resources becoming available in order to support the formidable body of knowledge, known as Parallel Random Access Model (PRAM) algorithmics [7], and the latent, though not widespread, familiarity with it. Driven by the repeated programming difficulties of parallel machines, ease-of-programming was a leading design objective of XMT.

The XMT architecture, depicted in Figure 1, includes an array of lightweight cores, Thread Control Units (TCUs), and a serial core with its own cache (Master TCU). The architecture

includes several clusters of TCUs connected to mutually-exclusive shared cache modules by a highly optimized interconnection network [1]. XMT does not feature writable private caches except for the Master TCU. Moreover, since the cache modules are mutually exclusive, no cache coherence is required. TCUs include lightweight ALUs, but the more heavy-weight units are shared by all TCUs in a cluster. XMT is programmed in XMTC, a simple extension of the C language which contains succession of serial and parallel code sections. The code of a parallel section is expressed in the SPMD (single program, multiple data) style, specifying an arbitrary number of virtual threads sharing the same code. Further details on the XMT architecture can be found in [10].

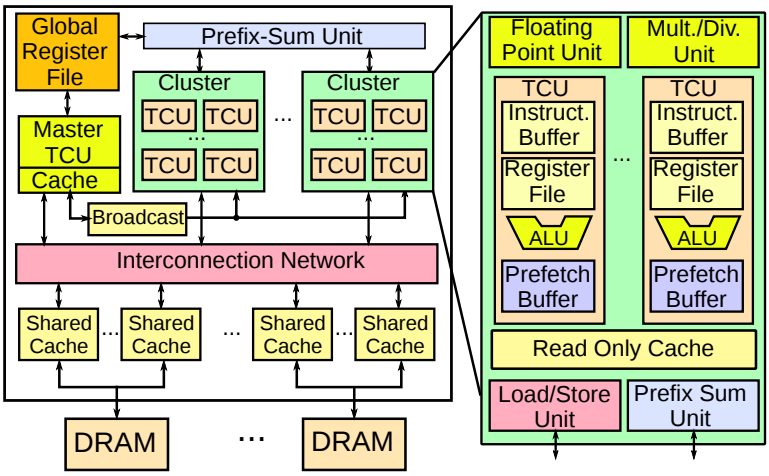

Figure 1: *The XMT architecture.*

XMT allows concurrent instantiation of as many threads as the number of available processors. Tasks are efficiently started and distributed thanks to the use of prefix-sum for fast dynamic allocation of work and a dedicated instruction and data broadcast bus.

To handle a high level of parallelism, the memory architecture of XMT partitions data from the first level of on-chip caches. The interconnection network (ICN) connects the TCUs (processors) and the shared caches. It is implemented in a Mesh-of-Trees configuration with the goal of minimizing the number of queuing bottlenecks for fast and reliable communication [2].

Architectural Simulation of XMT

Computer architects rely on software simulation tools to model and evaluate future architectures. The main advantage of simulation is that it is less costly and more flexible than hardware implementation. A main disadvantage is that software is significantly slower than hardware, which inhibits the extent of evaluation that is feasible in simulation. For instance, it may take hours of simulation time to model one second of hardware processing time.

XMTSim is the highly configurable cycle-accurate simulator of the XMT architecture. It accurately models the interaction between micro-architectural components, including the TCUs, functional units, caches, and the interconnection network. XMTSim is verified against the

64-TCU FPGA prototype of the XMT architecture. More details on the XMT toolchain can be found in [6].

In contrast with the majority of today's many-core processors, which may be utilized for a range of tasks, but are designed specifically for graphics processing [8], XMT is a general-purpose many-core processor by design. Establishing that XMT is a self-simulating machine would provide supporting evidence for its general-purposeness.

Demonstrating that XMT is a self-simulating machine can be done by simulating XMTSim on XMTSim. For completeness, XMTSim needs to be parallelized and efficiently run on the target XMT model. Parallelizing XMTSim also has the secondary benefit of improving simulation time: XMT simulations can be run on the 64-core FPGA computer and furthermore, the parallelized code can be used as a template to port XMTSim to current commercially available parallel machines.

Parallel ICN: Student-led Projects

The interconnect simulation constitutes a significant part of the full chip simulation time in the cycle-accurate XMT simulator, XMTSim. We found through profiling the serial-Java XMTSim on an Intel XEON server processor that it takes from 38% to 58% of the simulation time (depending on the program being simulated). Parallelizing the ICN is therefore a logical first step towards our overall objective of parallelizing XMTSim.

Undergraduate students at Swarthmore College, like in many other institutions, have limited time that they can dedicate for active research. Typically, they can choose to dedicate 10 weeks during a summer to full time research. Moreover, most students prefer to explore different research and internship opportunities during their undergraduate years, and as a result do not return to work on the same project for more than one summer.

The students that join our research are typically rising juniors or seniors, with some background in programming and digital systems, but with limited to no background in computer architecture and parallel and distributed systems. We therefore must utilize those 10 weeks efficiently to familiarize the students with our research project, train them to use the XMT toolchain, make progress, and document their work for future students. The effort dedicated by students to documentation and its quality has a significant influence on the experience and productivity of future students.

Any student joining our team should ideally be able to start with a well-defined task that can then be expanded if time permits. The ICN code of XMTSim is self-contained and more basic than other components of the simulator, making it more suitable for smaller projects, and for a limited time scope. Next, we describe the ICN simulation and the scope of past student projects.

When we search for students to join our group, we give preference to those students that have better preparation and demonstrate more profound interest in our work. On top of these criteria, we also give preference to female and minority students. While our research does not offer aspects that may target these students in particular, we aim to encourage these students to work with us.

ICN Simulation

The Mesh-of-Trees Interconnection Network (MoT-ICN) is composed of fan-in and fan-out trees. There is a fan-out tree for each XMT processor cluster sending memory requests, with the processor cluster representing the root of the tree, and a fan-in tree for every memory module receiving requests, with the memory module representing the root of the tree. A similar network returns data from memory to the processor clusters. A 64-core XMT MoT-ICN consists of two sets of 64 fan-in and 64 fan-out trees to connect 64 cores with 64 memory modules in both directions. Sample trees are shown in Figure 2.

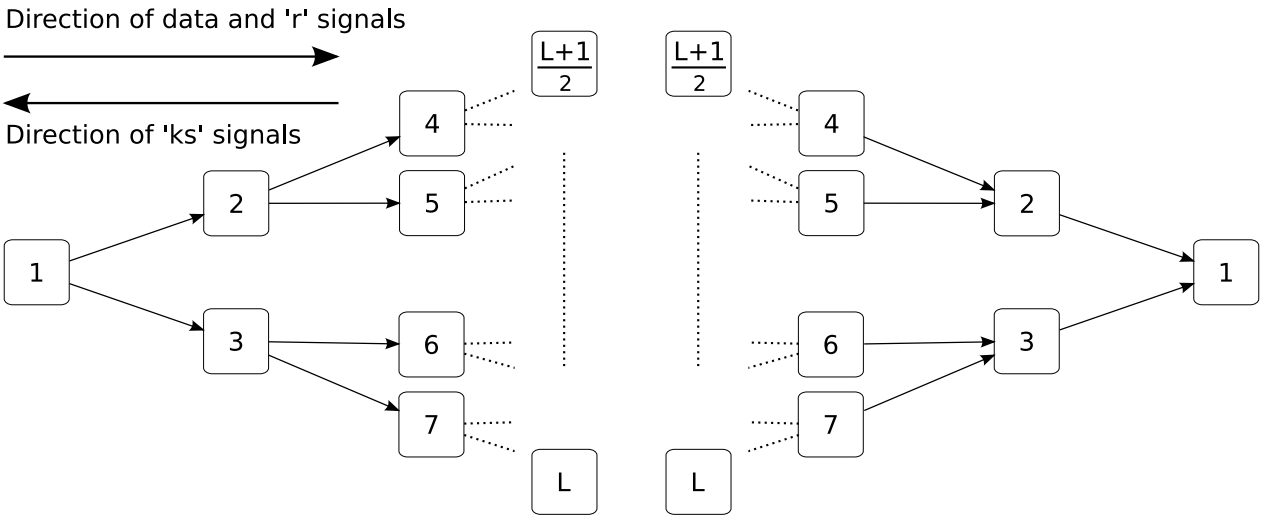

Figure 2: *Sample ICN fan-out and fan-in trees.*

The basic building block of each MoT-ICN tree is a pipeline primitive, representing a node in the tree. The pipeline primitive is simulated as 2-entry FIFO, as in Figure 3. The modeling of each stage requires two data storage holders, which we call IN and OUT in reference to the input and output buffers of a FIFO, and two signal variables, *r* and *ks*. These signals communicate the occupation and availability states between neighboring FIFOs. Simulation of one clock cycle consists of *step* and *reset* phases. In the step phase, the data is moved between stages based on the values of *r* and *ks*. In the *reset phase* the values of the *r* and *ks* signals are set according to the number of full buffers. In the *shift phase*, the data is moved from the IN to the OUT buffers for FIFOs that contain a single job. Within each of these phases, the simulation can potentially be performed in parallel for all pipeline stages. However, correctness becomes an issue when the actions of phases are intermixed.

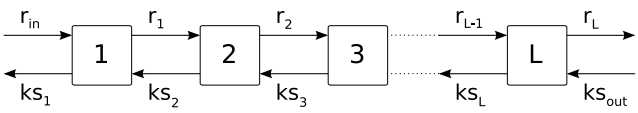

Figure 3: *Sample pipeline. Only ks and r signals are shown, and data flows from left to right.*

In order to evaluate the effectiveness of a parallel simulation, we need to generate an optimized serial simulation of the MoT-ICN to serve as a baseline. The simulation of a full ICN consists of the following steps:

1. Run the reset phase of all trees.
2. Determine the tree input ports for which packets will be generated (for setting the *r* signals at the tree roots).
3. Run the step phase of all trees.
4. Pass the outputs of fan-out trees to fan-in trees.
5. Collect outputs.
6. Set input packets.

Project I

The first undergraduate student project involved optimizing the serial version of the ICN simulation, as well as making an initial attempt to parallelize it with different levels of granularity.

The simulation of the MoT-ICN can be programmed with different degrees of parallelism:

- Serial - The entire network is assigned to a single thread.
- Course-grained parallelism - Each tree is assigned to a dedicated thread.
- Medium-grained parallelism - Groups of nodes in each tree are assigned to a dedicated thread.
- Fine-grained parallelism - Each node is assigned to a dedicated thread.

In the serial version we can control the order in which the nodes are processed. This allows certain optimizations that overlap the actions in different phases, hence minimizing the total number of operations.

The fine-grained parallel version provides maximum parallelism, since every node in the network can be updated simultaneously, using a separate thread. However, the order in which the nodes are processed is not deterministic when this level of parallelism is introduced, thus preventing us from utilizing some of the optimizations used in the serial version.

With a coarse-grained parallel version, one tree is allocated to each thread. As a result, there are no conflicts between threads, and the same optimizations used in the serial version can be applied. However, this version limits the number of threads, and under-utilizes the processors on a 1024-core XMT configuration.

A medium-grained parallel version allocates multiple threads per tree. An individual tree can either be split up by node number (Figure 4) or by “sub-trees” (Figure 5). The former allocation technique allows to parameterize the number of nodes assigned to each thread for optimizing performance, but introduces conflicts that prevent using the serial version optimizations. The latter allocation technique makes for a more complicated assignment of threads to nodes, but

guarantees no conflicts between internal nodes of sub-trees, thus allowing for more code optimizations.

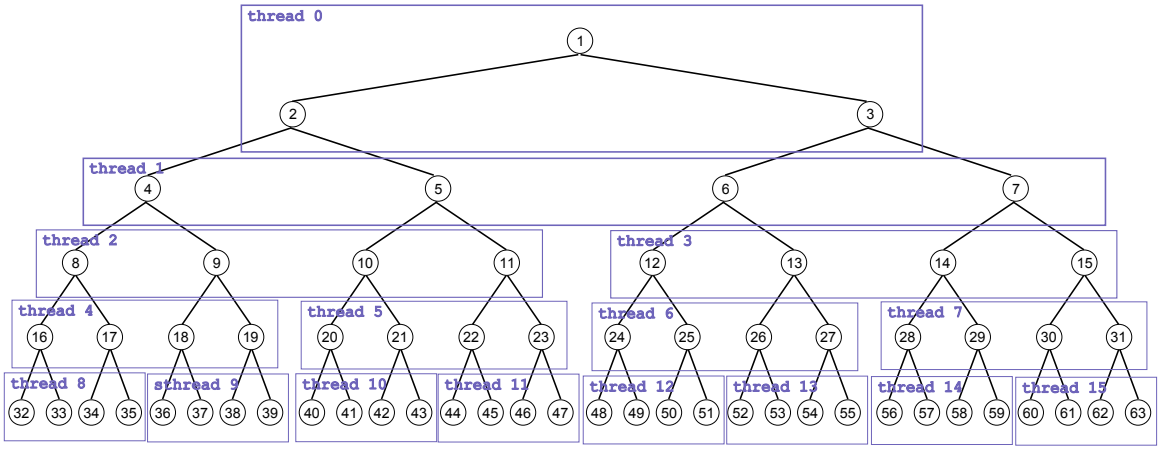

Figure 4: *Allocating threads based on node number.*

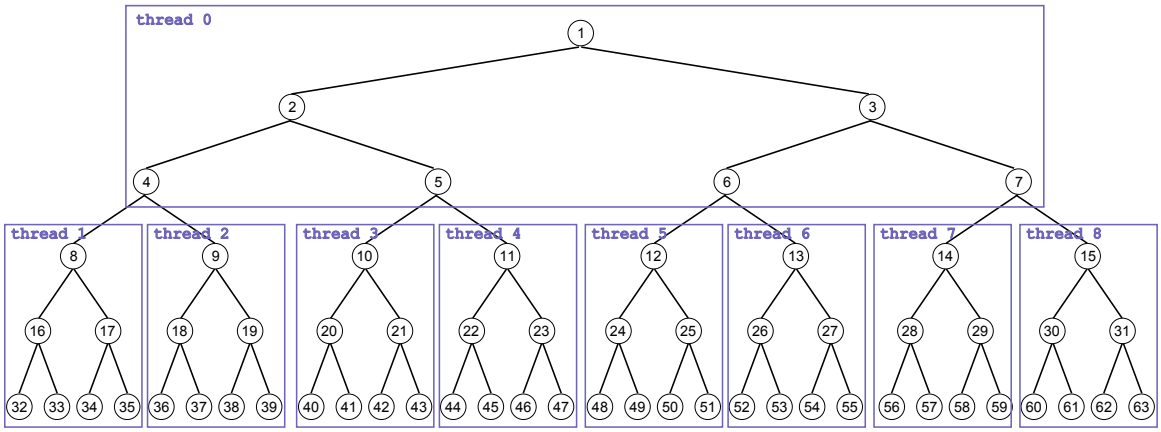

Figure 5: *Allocating threads based on subtrees.*

Project II

The baseline serial and subsequent parallel simulations described so far iterate through all the nodes in the tree in each of the simulation phases. We call this approach the *every-node* simulation. In order to improve simulation speed, we propose a fundamental change in the underlying simulation algorithm, called *active node* simulation, where we only iterate through those nodes in the tree that contain data packets at any given simulation phase. This approach is potentially more efficient when the number of packets in the ICN is significantly lower than the number of nodes.

The second undergraduate student project involved modifying both the serial and best performing parallel version of the ICN simulation to implement the *active-node* algorithm, and comparing the two. Currently, only the step phase loops through the active nodes and the reset phase is used to create an array containing references to the active nodes.

Experimental Results

We simulated a 64 processor, 64 memory module ICN configuration on a 64-core XMTSim platform. Figure 6 presents the highlights of our results.

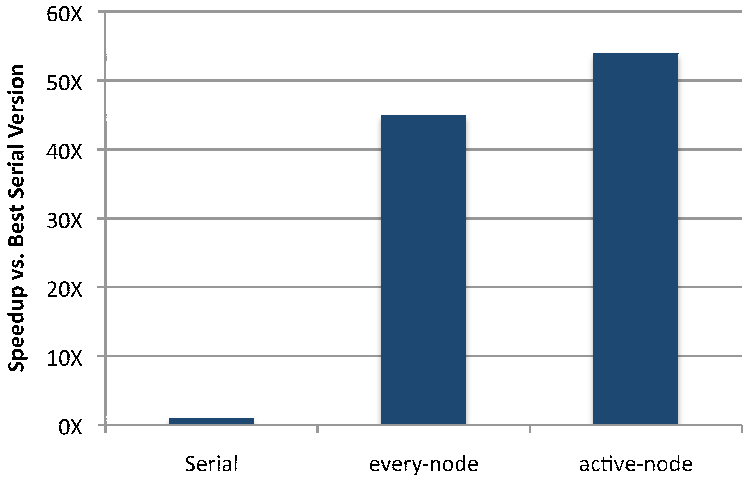

Figure 6: *Speedup vs. best serial version.*

The every-node approach was simulated with each of the four variants: serial execution, coarse, medium, and fine-grained parallelism. The best serial execution was run on the embedded serial processor of XMT (not the Intel processor mentioned earlier). The coarse-grained parallelism version was the optimal one, with speedups of 45x over the best serial version. As discussed above, this version allocates one tree to each thread and thus minimizes synchronization conflicts between threads. The simulated ICN consists of 2x64 trees, which scale well with our simulated 64-core XMTSim platform. The other versions demonstrate lower performance due to the overhead incurred by instantiating a larger number of threads.

The active-node approach achieves additional gains, providing a speedups of up to 54x compared to the best serial implementation. These speedups are achieved with the coarse-grained parallelism variant, with a network load where one packet is sent through each fan-out tree root every cycle. These packets are then routed homogeneously along the available destination ports. Recall that the active-node approach only iterates through tree nodes containing data packets at any point during simulation. We simulated this approach with variable network load, and it demonstrates increasing speedups as the number of packets in the network decreases, resulting in an increasing number of inactive tree nodes.

It is important to note that active-node speedups are obtained in the step phase alone, where the optimization effort was concentrated. In fact, some of the speedups are offset by a slight slowdown in the reset phase, due to additional overhead required for the creation of the active-node data structures. Future work will focus on parallelizing the reset phase.

Educational Insights

This is a pilot study of an on-going multi-year project. As such, validating and measuring our success in achieving our stated goals is a continuing process. First, the fact that the students have been able to provide correct (e.g., working) programs with speedups over serial versions is a significant form of validation of the teachability of our XMT platform. The results demonstrate a step towards the completion of our combined milestone of a parallel ICN. Furthermore, both students contributed to the selection and definition of the work to be performed. In order to do this effectively, they had to highly familiarize themselves with the project as a whole.

What makes our project distinct from other research projects is the adaptation of an active long-term research project to a set of limited scope projects to be done solely by undergraduate students. To this end, we seek to define individual projects whose overall long term result is greater than the sum of the parts. This, along with the limited time-scope of each project, is different from other research projects designed specifically for undergraduate students. For example, Hadfield describes the integration of research experience into the undergraduate curriculum [4]. In their case, research training is provided to the students over multiple years, as they work towards a cumulative research experience. In contrast, we aim to make the best out of a single 10 week period that each student has available to dedicate to this research project.

Validating the objective of motivating undergraduate engineering students, in our case women, to pursue an engineering career path is more complex. In order to effectively access this objective, we would need to continue the project for a few more years, have more undergraduate students participants, and also be able to follow the participants post graduation. As a short-term feedback we interviewed the two students involved in the project thus far.

Both students, K and R, felt that their summer research experience had some contribution in re-enforcing their future plans for a career in engineering.

K was able to obtain a summer internship in the following summer. She found that her summer research experience was very helpful in her work since, in contrast to other interns, she did not feel overwhelmed by entering a large coding project. As part of our research team, she was able to gain meaningful experience in a larger scope project, in familiarizing herself with code written by someone other than herself, and in modifying and extending it coherently with its current programming style.

R also felt that her experience taught her a lot about organization and design of a large programming project using modularity. She thought one of the important things she learned from this experience was learning to study independently from available resources. Since she had no background in computer architecture before joining the project, she spent part of the summer learning architecture, as well as strengthening her background in C programming.

R found that her involvement in this project encouraged her to pursue a graduate degree in computer engineering after graduation, and also helped her determine which type of research she would specifically be interested in pursuing. It also opened up more options for her next summer, since she now feels confident about being able to obtain both other summer research positions and internships. K plans to obtain a job in the computer engineering field following graduation, and recommended our research project to R. Both said they would recommend it to others.

Conclusion

The XMT is a novel many-core architecture which is well suited for teaching the challenging task of writing parallel programs in a many-core environment. This project presented a great opportunity for undergraduate students to participate in cutting-edge research while reinforcing their choice of pursuing an engineering career.

While undergraduates realize that their work cannot achieve a high-impact research goal alone, our students were able to provide a validated first step towards the completion of our goal of self-simulation. The experience they gained encouraged them to explore additional engineering internships and research experience prior to graduation.

As future work, we intend to continue this project with more student participants. Once more data is available, we intend to provide a more extensive evaluation of our goals. We are also interested in comparing the career paths pursued by students involved in this project to those pursued by students involved in other research projects and those not participating in active research during their undergraduate education.

Acknowledgment

Partial support by NSF grants , 0926237, 0811504 and 0834373 is gratefully acknowledged.

References

[1] A. O. Balkan, M. N. Horak, G. Qu, and U. Vishkin, "Layout-accurate design and implementation of a high-throughput interconnection network for single-chip parallel processing," in *Proceedings of Hot Interconnects*, 2007.

[2] A. O. Balkan, G. Qu, and U. Vishkin, "A mesh-of-trees interconnection network for single-chip parallel processing," in *Proceedings of the IEEE 17th International Conference on Application-specific Systems, Architectures and Processors (ASAP'06)*, Steamboat Springs, Colorado, September 2006, pp. 73–80.

[3] C. R. A. W. CRAW, "DREU: Distributed research experiences for undergraduates," 2011. [Online]. Available: https://parasol.tamu.edu/dreu/

[4] S. Hadfield and D. Schweitzer, "Building an undergraduate computer science research experience," in *Proceedings of the 39th IEEE international conference on Frontiers in education conference*, ser. FIE'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 1193–1198. [Online]. Available: http://dl.acm.org/citation.cfm?id=1733663.1733942

[5] M. Herlihy and N. Shavit, *The Art of Multiprocessor Programming*. Morgan Kaufmann, 2008.

[6] F. Keceli, A. Tzannes, G. Caragea, U. Vishkin, and R. Barua, "Toolchain for programming, simulating and studying the XMT many-core architecture," in *Proceedings of the International Workshop on High-Level Parallel Programming Models and Supportive Environments*, 2011, in conj. with IPDPS.

[7] J. Keller, C. Kessler, and J. L. Traeff, *Practical PRAM Programming*. New York, NY, USA: John Wiley & Sons, Inc., 2001.

[8] J. Nickolls and W. Dally, "The GPU computing era," *IEEE Micro*, vol. 30, no. 2, pp. 56 –69, March 2010.

[9] U. Vishkin, "Using simple abstraction to guide the reinvention of computing for parallelism," *Communications of the ACM*, vol. 54, no. 1, pp. 75–85, Jan. 2011.

[10] X. Wen and U. Vishkin, "FPGA-based prototype of a PRAM on-chip processor," in *Proceedings of the ACM Computing Frontiers*, 2008.