UNICORN: A BULK SYNCHRONOUS PROGRAMMING MODEL, FRAMEWORK AND RUNTIME FOR HYBRID CPU-GPU CLUSTERS

TARUN BERI

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY DELHI
SEPTEMBER 2016

UNICORN: A BULK SYNCHRONOUS PROGRAMMING MODEL, FRAMEWORK AND RUNTIME FOR HYBRID CPU-GPU CLUSTERS

by

TARUN BERI

Department of Computer Science and Engineering

Submitted
in fulfillment of the requirements of the degree of Doctor of Philosophy
to the

INDIAN INSTITUTE OF TECHNOLOGY DELHI
SEPTEMBER 2016

Dedicated to my Family, Teachers and Friends ...

Certificate

This is to certify that the thesis titled **Unicorn: A Bulk Synchronous Programming Model, Framework and Runtime for Hybrid CPU-GPU Clusters** being submitted by **Mr. Tarun Beri** to the **Indian Institute of Technology Delhi** for the award of the degree of **Doctor of Philosophy** in Computer Science and Engineering is a record of original bonafide research work carried out by him under our supervision. The work presented in this thesis has reached the requisite standard and has not been submitted elsewhere either in part or full for the award of any other degree or diploma.

Dr. Subodh Kumar
Associate Professor
Computer Science and Engineering
Indian Institute of Technology Delhi

Dr. Sorav Bansal
Assistant Professor
Computer Science and Engineering
Indian Institute of Technology Delhi

Acknowledgements

I always knew doing a Ph.D. from India's premier institute is tough and along with a full time corporate job it was going to be even more so. I am so grateful to my guide *Prof. Subodh Kumar* and co-guide *Prof. Sorav Bansal*, who have understood my dual responsibilities at all times and made this journey so smooth that today when I look back, it has actually been an amazing experience. Today, I hold the highest regard for both of them and I do not even have the slightest hesitation in recommending them as Ph.D. guides to anyone. I am actually feeling uncomfortable of not being able to put their invaluable contributions into appropriate words. Throughout the duration of my Ph.D., their prudent advice and insightful knowledge have been a constant source of inspiration and encouragement. Prof. Subodh's surprise visit to our paper presentation at IPDPS 2015 was quite reassuring as well. I still remember the time when our paper faced a second consecutive reject in a Tier 1 conference and both of them helped me sail through that tough moment. To quote them "Many good papers get rejected 3 to 4 times before they really rise to the top". I am still learning a lot of things from them and hope to replicate their formal writing skills one day.

My sincere thanks also goes to *Prof. Kolin Paul*, *Prof. Preeti Panda* and *Dr. Yogish Sabharwal* who have been on the review committee for 5+ years and have constantly provided invaluable guidance and feedback.

Working extremely hard and not stumbling to any challenge or work pressure was infused in me right from my childhood. Being born to an influential and highly educated joint family in a small town was a big boon. Somehow it

taught me to learn from each and every incident and person I came across. My grandparents, parents, paternal and maternal uncles, cousins, in-laws and their families were all instrumental in defining this success. When I was a kid, everyone in my family especially my grand parents *Mr. Nitrajan Pal Beri* and *Mrs. Shakuntla Beri*, my father *Mr. Arun Beri* and my uncles *Mr. Anil Beri* and *Mr. Raman Beri* taught me for long hours and helped me build my understanding on several basic concepts. My mother, *Prof. Chander Beri*, who herself is a Ph.D. in mathematics, is the biggest motivation that has driven me this far. It was her wish that I join this Ph.D. program and its her confidence in me that I am now seeing its completion. Getting married during the Ph.D. could have complicated things but my understanding and supporting partner *Tanvi* and her parents *Dr. Rajinder Mehta* and *Mrs. Mala Mehta* turned it into a delightful journey. Birth of my son, *Taanish*, during this tenure proved to be the much needed rejuvenation. My brother *Ankur* and his wife *Ria* also deserve a very special mention. Several of my responsibilities often got offloaded to them while I was busy with my dual work. Without any grief, they happily accepted this. God has also been very kind to me.

Today, while writing this acknowledgement, I also recall all my school and college teachers who have time and again instilled confidence in me and made me believe in myself. I would also like to take this opportunity to thank my long time friends *Vineet Jindal*, *Jitesh Saghotra*, *Gurpreet Bedi*, *Amandeep Bawa*, *Vikas Sarmal*, *Ajay Paul Singh*, *Amit Goyal*, *Gagan Bansal*, *Prateek Prashar*, *Bikramjit Singh*, *Anand Sinha*, *Vijay Sharma*, *Asif Iqbal*, *Anish Singla*, *Gaurav Gupta*, *Dilraj Singh*, *Thalinder Singh*, *Gagandeep Singh* and

Narayan who have selflessly stayed along throughout.

My colleagues and friends at work *Vineet Batra*, *Avinandan Sengupta*, *Ravi Ahuja*, *Inderpal Bawa*, *Jatin Sasan*, *Harish Kumar*, *Tariq Rafiq*, *Sachin Patidar*, *Rohil Sinha*, *Vaibhav Tyagi*, *Vikas Marda*, *Gordon Dow*, *Ahsan Jafri* and *Pushp Agarwal* also deserve a very special recognition for supporting me helping me out directly or indirectly. While a few spared time to discuss my project and share their feedback and new ideas, others helped me grasp new technological advancements and learnings. I would also like to thank the management of Adobe Systems India Pvt. Ltd. (*Pankaj Mathur*, *Rajesh Budhiraja*, *Lekhraj Sharma*, *Gaurav Jain* and *Viraj Chatterjee*), Cadence Design Systems India Pvt. Ltd. (*Utpal Bhattacharya* and *Parag Chaudhary*) and STMicroelectronics India Pvt. Ltd. (*Anand Singh* and *Vivek Sharma*) who have supported me and let me pursue my studies along with the office work.

I would also like to thank M.Tech. students *Harmeet Singh*, *Vigya Sharma*, *Abhishek Raj*, *Abhishek Kumar* and *Ameen Mohammed* who have worked with me and helped develop the ideas in this work. In the end, my sincere thanks goes to my mother's friend and my teacher *Mrs. Savita Sharma* and her husband *Mr. Sunil Sharma* who have always encouraged and supported me since my childhood. I hope I shall make everyone proud in the time to come and work even harder to take this work and knowledge to the next level.

Tarun Beri

Abstract

Rapid evolution of graphics processing units (GPUs) into general purpose computing devices has made them vital to high performance computing clusters. These computing environments consist of multiple nodes connected by a high speed network such as Infiniband, with each node comprising several multi-core processors and several many-core accelerators. The difficulty of programming hybrid CPU-GPU clusters often limits software's exploitation of full computational power. This thesis addresses this difficulty and presents ***Unicorn*** – a novel parallel programming model for hybrid CPU-GPU clusters and the design and implementation of its runtime.

In particular, this thesis proves that efficient distributed shared memory style programing is possible. We also prove that the simplicity of shared memory style programming can be retained across CPUs and GPUs in a cluster, minus the frustration of dealing with race conditions. And this can be done with a unified abstraction, avoiding much of the complication of dealing with hybrid architectures. This is achieved with the help of transactional semantics, deferred bulk data synchronization, subtask pipelining and various communication and computation scheduling optimizations.

Unicorn provides a bulk synchronous programming model with a global address space. It schedules concurrent tasks of a program in an architecture and topology oblivious manner. It hides the network and exposes CPUs and accelerators loosely as bulk synchronous computing units with logical phases, respectively, of local computation and communication. Each task is further decomposed into coarse-grained concurrently executable subtasks that Unicorn schedules transparently on to available CPU and GPU devices in the cluster. Subtasks employ transactional memory semantics to access and synchronize data, i.e., they check out a private view of the global shared memory

before their local computation phase and check in to the global shared memory afterwards, optionally resolving conflicting writes in a reduction step.

Unicorn's main design goals are easy programmability and a deterministic parallel execution environment. Device, node and cluster management are completely handled by the runtime and no such API is exposed to the application programmer. Load balancing, scheduling and scalability are also fully transparent to the application code. Application programs do not change from cluster to cluster to maintain efficiency. Rather, Unicorn adapts the execution to the set of present devices, the network and their dynamic load. Application code is oblivious to data placement within the cluster as well as to changes in network interfaces and data availability pattern. *Unicorn's* programming model, being deterministic, eliminates data races and deadlocks.

To provide efficiency, *Unicorn's* runtime employs several optimizations. These include prefetching task data and pipelining subtasks in order to overlap their communication with computations. *Unicorn* employs pipelining at two levels – firstly to hide data transfer costs among cluster nodes and secondly to hide DMA communication costs between CPUs and GPUs on all nodes. Among other optimizations, *Unicorn's* work-stealing based scheduler employs a two-level victim selection technique to reduce the overhead of steal operations. Further, it employs special proactive and aggressive stealing mechanism to prevent the said pipelines from stalling (during a steal operation). To prevent a subtask (running on a slow device or on a device behind a slow network or I/O link) from becoming a bottleneck for the entire task, *Unicorn* reassesses its scheduling decisions at runtime and schedules a duplicate instance of a straggling subtask on a potentially faster device. *Unicorn* also employs a software LRU cache at every GPU in the cluster to prevent the shared data between subtasks getting DMA'ed more than once. To further boost GPU performance, *Unicorn* makes aggressive use of CUDA streams and schedules multiple subtasks for simultaneous execution.

To evaluate the design and implementation of *Unicorn*, we parallelize several coarse-grained scientific workloads using Unicorn. We study the scalability

and performance of these benchmarks and also the response of *Unicorn's* runtime by putting it under stress tests like changing the input data availability of these experiments. We also study the load balancing achieved in these experiments and the amount of time the runtime spends in communications.

We find that parallelization of coarse-grained applications like matrix multiplication or 2D FFT using our system requires only about 30 lines of C code to set up the runtime. The rest of the application code is regular single CPU/GPU implementation. This indicates the ease of extending sequential code to a parallel environment. The execution is efficient as well. Using GPUs only, when multiplying two square matrices of size $65536 * 65536$, *Unicorn* achieves a peak performance of 7.81 TFlop/s when run over 28 Tesla M2070 GPUs (1.03 TFlop/s theoretical peak) of our 14-node cluster (with subtasks of size $4096 * 4096$). On the other hand, CUPLAPACK [28], a linear algebra package specifically coded and optimized from scratch, reports 8 TFlop/s while multiplying two square matrices of size $62000 * 62000$ using 32 Quadro FX 5800 GPUs (0.624 TFlop/s theoretical peak) of a 16 node cluster connected via QDR InfiniBand.

Fine-grained applications, however, may not fit into our system as efficiently. Such applications often require frequent communication of small data. This is inherently against our bulk synchronous design and more advanced optimizations may be needed to make these applications profitable.

Contents

1 Introduction **1**

1.1 Application Model 3

1.2 Global Address Space 4

1.3 Scheduling 5

1.3.1 Reducing Steal Overhead 7

1.3.2 Handling CPU-GPU Performance Disparity 7

1.4 Performance Optimizations 9

1.4.1 Data Transfer Optimizations 10

1.4.1.1 Locality Aware Scheduling 10

1.4.1.2 Pipelining 11

1.4.1.3 Grouping Communications 11

1.4.1.4 Software GPU Caches 12

1.4.2 Scheduling Optimizations 12

1.4.3 Address Space Optimizations 13

1.5 High Level Abstractions 13

1.6 Epilogue 14

2 Programming Model **17**

2.1 Data Subscriptions and Lazy Memory 23

3 Runtime System **25**

3.1 Device and Node Management 25

3.1.1 Runtime Internals 27

3.1.2 List of Threads 31

3.2 Shared Address Spaces 32

3.3 Network Subsystem 41

3.4 Pipelining 44

3.5 Scheduling Subsystem 45

3.5.1 Work Stealing in Unicorn 46

3.5.1.1 ProSteal 49

3.5.1.2 Locality-aware Scheduling 52

3.5.1.2.1 Greedy Scheduling 54

3.5.1.2.2 Locality-aware work stealing . . . 55

3.5.2 Work-Group Calibration 55

3.5.3 Multi-Assign 57

3.5.3.1 Subtask Cancellation 58

3.5.4 Scheduling Across Task Barriers 59

3.5.5 Scheduling Concurrent Tasks 60

3.6 Software GPU Cache 60

3.7 Conflict Resolution 61

4 Pseudo Code Samples **65**

5 Experimental Evaluation 75

5.1 Unicorn Parallelization of Benchmarks 78

5.1.1 Characteristics of Benchmarks 81

5.2 Performance Scaling 82

5.2.1 CPU versus GPU versus CPU+GPU 85

5.2.2 PageRank 88

5.3 Scheduling 89

5.3.1 Work Stealing 90

5.3.1.1 One-level vs. Two-level 91

5.3.1.2 ProSteal 94

5.3.2 Locality-aware Scheduling 94

5.4 Load Balancing 100

5.5 Stress Tests 102

5.5.1 Heterogeneous Subtasks 103

5.5.2 Input Data Distributions 103

5.5.3 Varying Subtask Size 104

5.6 Unicorn Optimizations 105

5.6.1 Multi-Assign 106

5.6.2 Pipelining 108

5.6.3 Software cache for GPUs 109

5.6.4 Data Compression 110

5.7 Overhead Analysis 111
5.7.1 Varying CPU cores 112
5.7.2 Unicorn Time versus Application Time 114
5.7.3 Data Transfer Frequency 115
5.8 Unicorn versus others 116

6 Application Profiling **119**

7 Public API **125**

8 Related Work **145**

9 Conclusions and Future Work **151**

Bibliography **153**

Appendix **163**

10.1 Unicorn's MapReduce Extension 163
10.2 Scratch Buffers 164
10.3 Matrix Multiplication Source Code 165

Unicorn Publications **175**

Biography **177**

List of Illustrations

Figures

2.1 Application program in Unicorn 18

2.2 Mapping a BSP superstep to a Unicorn task 19

2.3 Execution Stages of a Subtask 20

2.4 Hierarchical Reduction - leaf nodes are subtasks, others are reduced subtasks; dotted lines are inter-node data transfer . . 21

3.1 The design of the Unicorn runtime – Light orange region represents the instance of the runtime on each node in the cluster 27

3.2 Address Space Ownerships – PD represents Address Space Ownership Directory, TD represents Temporary Ownership Directory, x represents non-existent directory, red text represents changes from last state and cells in light blue background represent directory changes via explicit ownership update messages; Node 1 is the address space master node. 37

3.3 Loss in victim's pipeline due to work stealing 50

5.1 Characteristics of various benchmarks 82

5.2 Performance analysis of various benchmarks 83

5.3 Scaling with increasing problem size 85

5.4 Image Convolution – GPU vs. CPU+GPU 86

5.5 Matrix Multiplication – GPU vs. CPU+GPU 86

5.6 2D FFT – GPU vs. CPU+GPU 87

5.7 Experiments with matrices of size 32768×32768 (lower is better) 88

5.8 Page Rank 89

5.9 Centralized scheduling versus Unicorn – 14 nodes 90

5.10 One-level versus two-level work stealing 92

5.11 Work Stealing – with and without ProSteal 93

5.12 Locality aware scheduling 96

5.13 Locality aware scheduling (Contd.) 97

5.14 Image Convolution: Locality aware work-stealing (*Derived Affinity*) 99

5.15 Matrix Multiplication: Locality aware work-stealing (*Derived Affinity*) 99

5.16 Load Balancing (Image Convolution) – W denotes a CPU work group and G denotes a GPU device – Block random data distribution 101

5.17 Load Balancing (Page Rank) – W denotes a CPU work group and G denotes a GPU device 102

5.18 Load Balancing (Matrix Multiplication) – 32768×32768 matrices – Centralized data distribution 102

5.19 Load Balancing (Heterogeneous Subtasks) – W denotes a CPU work group and G denotes a GPU device 103

5.20 Impact of initial data distribution pattern 104

5.21 Subtask size ($N \times N$) – experiments executed on 14 nodes . . 105

5.22 Multi-Assign (no external load) 106

5.23 Multi-assign under external load (Image Convolution) – 4 nodes106

5.24 Pipelining (Image Convolution) 108

5.25 Matrix Multiplication – GPU Cache Eviction Strategies 109

5.26 PageRank data compression (250 million web pages) 111

5.27 Varying CPU cores used in subtask computation 112

5.28 Matrix Multiplication – Varying CPU core affinity 113

5.29 Library time versus application time 114

5.30 Data Transfer Frequency 115

5.31 Unicorn versus StarPU 117

5.32 Unicorn versus SUMMA 117

6.1 Sample Unicorn Logs (part 1) 119

6.2 Sample Unicorn Logs (part 2) 120

6.3 Sample Unicorn Logs (part 3) 121

6.4 Performance 123

6.5 Load Balance 123

6.6 Compute Communication Overlap 124

6.7 Event Timeline 124

Codes

4.1 Unicorn program for square matrix multiplication 66

4.2 Unicorn callbacks for square matrix multiplication 68

4.3 Unicorn program for 2D-FFT 70

4.4 Unicorn callbacks for 2D-FFT 72

7.1 Unicorn Header: pmPublicDefinitions.h 125

7.2 Unicorn Header: pmPublicUtilities.h 142

10.1 File matmul.h 165

10.2 File matmul.cpp 166

10.3 File matmul.cu 172