A Temporally Reconfigurable Multi-Accelerator Parallel Architecture for Reuse and Throughput Oriented Computing

Masab Ahmad, Halit Dogan, and Omer Khan
University of Connecticut, Storrs, CT, USA
{masab.ahmad, halit.dogan, khan}@uconn.edu

ABSTRACT

Machine learning and graph applications are highly heterogeneous in the algorithmic and input data behaviors, leading to performance variations in today's parallel accelerator architectures. Various parallel machines target such applications with either throughput computing using GPU's massive threading, or exploit data reuse using strong multicore cores and on-chip caching of data. There are ample benefits in both architectures exploiting parallelism, throughput, and data reuse. However, a single machine architecture has not been explored to exploit both throughout and reuse capabilities in a temporally reconfigurable setup. This work builds a multi-accelerator parallel architecture that supports variable hardware threading capabilities to support reuse or throughput computing paradigms at runtime. The existing QUARQ architecture is extended with per-core fine-grain hardware multithreading support to enable the throughout mode of operation. However, the reuse mode is efficiently supported in the underlying QUARQ architecture using hardware cache coherence for efficient data movement on-chip, and a novel moving compute to data model for efficient synchronization between threads. Results show that the extended QUARQ architecture delivers the right performance situationally, i.e., between reuse and throughput modes of operation, the best performing mode is highly algorithm and input dependent for the target machine learning and graph processing domains.

1. INTRODUCTION

Target applications utilizing graph and machine learning have risen rapidly over the past decade [1] [2]. However, software variations in such benchmarks and inputs make it hard to fully exploit performance. Graph applications tend to require high throughput computing to allow processing of millions or even billions of vertices and edges with high performance. On the other hand, machine learning applications require floating point computations and benefit from high data reuse on chip. Such benchmark variations perform best using different architectures and concurrency selections that are either designed for throughput or reuse bound computing. Mapping a benchmark to the right architecture thus becomes challenging, as programmers need to tune concurrency choices. It has been shown that performance variations can be exploited if provided with a heterogeneous computing platform [3]. Applications with high data reuse, such as machine learning workloads perform better on machines with better single threaded floating point performance and stronger caching [4,5]. However, path planning graph workloads, such as the Bellman-Ford algorithm can be highly parallelized on throughput machines that offer thousands of concurrent threads (e.g., GPUs) [6]. As shown in Figure 1, this problem comprising benchmarks, inputs, and parallel implementations on heterogeneous architectures needs to be computed with near real-time efficiency. Thus, an architecture is desirable that exploits *both* throughput and reuse capabilities.

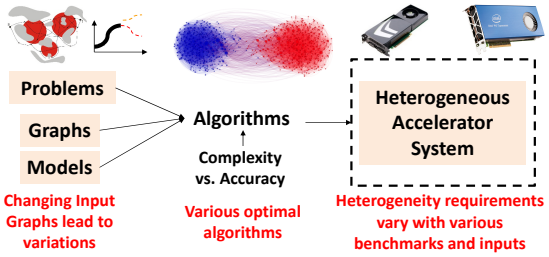

Figure 1: The Benchmark-Input-Implementation-Architecture Mapping Problem.

Today's single chip parallel machines utilize unary architecture types, e.g., Nvidia or AMD GPUs, or Intel multicores. GPU architectures have weaker on-chip caches but stronger threading and off-chip memory bandwidth capabilities, which allow for high throughput processing [7]. Large multicores, such as Intel's Xeon Phi expose less threading but offer strong cache hierarchies, hardware cache coherence, and atomic operations for synchronization to efficiently support on-chip data access and reuse. Our initial work integrated these two architectures *spatially* to develop a multi-accelerator setup for efficient graph processing [8, 9]. However, it is hard to utilize both architectures due to significant data movement overheads between multiple accelerators.

This paper proposes a *temporally* reconfigurable heterogeneous architecture that exposes both throughput and reuse computing capabilities to the programmer. The proposed architecture is based on the QUARQ multicore architecture, which is extended with fine-grain threading and floating-point SIMD capabilities to support throughput computing. In the baseline QUARQ architecture, programmability is supported using shared memory paradigm, and efficient data movement controls are supported using directory-based hardware cache coherence. Moreover, QUARQ features a novel moving compute to data model for efficient thread synchronization that is based on explicit messaging capabilities between cores [10, 11]. The proposed extension to QUARQ architecture incorporates variable threading with efficient register context swapping, and selective private cache bypassing capabilities. This allows the architecture to map a single thread per core for reuse mode, while map a variable number of

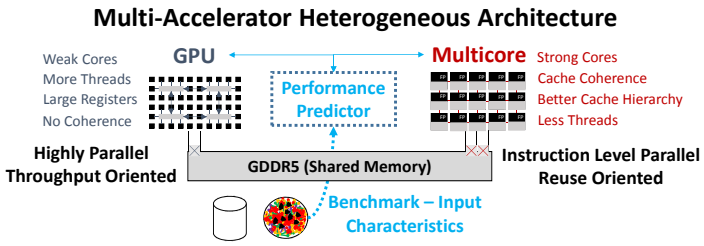

Figure 2: Spatial Incorporation of Throughput and Reuse oriented accelerators.

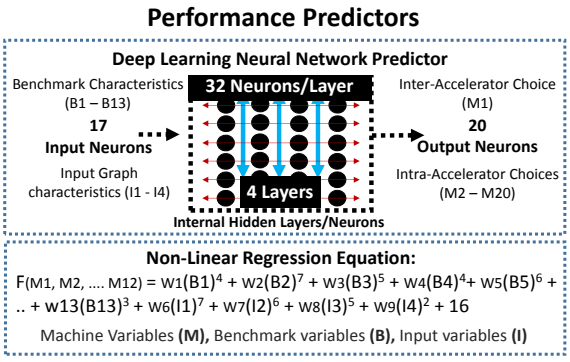

Figure 3: Performance Predictors for the Multi-accelerator Architecture.

threads per core to reconfigure the system in throughput computing mode. We envision the selection of throughput or reuse mode of operation using the novel performance predictor that our prior work explored in the context of *spatial* multi-accelerator architecture [8, 9]. Prototype analysis on graph and machine learning benchmarks show that better performance is achievable by selecting the right mode of operation for each benchmark. The achieved performance with the extended QUARQ architecture is shown to be competitive with state-of-the-art GPU and multicore machines.

2. MULTI-ACCELERATOR ARCHITECTURE

As throughput and reuse oriented machines exist individually, they can be spatially integrated to architect a heterogeneous setup. Our work, HeteroMap [9], integrates two machines (a GPU and a large scale multicore) on a common main memory (shown in Figure 2). The motivation for such a platform is that certain workloads perform well on GPUs, while others perform well on multicores. Graph workloads that are highly parallel benefit from the increased threading capabilities of GPUs, while machine learning workloads requiring floating point operations and data reuse benefit from multicores that implement hardware protocols for efficient on-chip data access.

A multi-accelerator platform exposes a humongous amount of concurrency choices within and across parallel machines. Therefore, scheduling for optimality in real-time situations becomes a challenging problem. HeteroMap selects concurrency choices as shown in Figure 2, where the performance predictor sits between the two machines and solves complex relationships to pick the right accelerator for the right benchmark-input combination. As shown in Figure 3, machine learning and regression models are utilized to reason about relationships between benchmark-input variables and accelerator parameters [8]. These models take in benchmark-input-machine characteristics, and predict architectural choices to be deployed for the multi-accelerator setup.

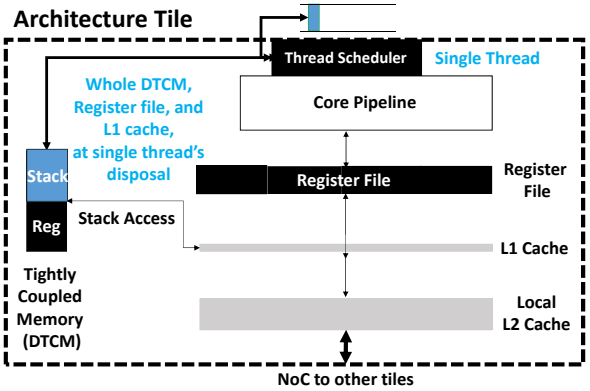

Figure 4: Proposed architecture for reuse mode of operation.

The spatial integration of multiple parallel machines in HeteroMap exposes the challenge of managing the data movement overheads between machines. Moreover, the aggregate utilization of the system is low in such a setup as both machines may not be utilized simultaneously. This is further magnified by the expanded form factor of the system in terms of area and power footprints. Acknowledging these challenges, this paper proposes a temporally reconfigurable multi-accelerator architecture that morphs between throughput and reuse computing modes.

3. EXTENDING QUARQ FOR REUSE AND THROUGHPUT COMPUTING

The proposed architecture is built on top of the QUARQ multicore architecture [10]. QUARQ is a single-chip tiled multicore architecture that is extensible to 1000-cores scale. The tiles are interconnected using a 2-D mesh network, and shared memory data access is supported using the scalable directory-based hardware cache coherence protocol. Moreover, QUARQ implements a novel moving compute to data model to accelerate thread synchronization. This synchronization model utilizes in-hardware explicit messaging between cores to move computations towards data, thereby improving synchronization by overlapping communication with other useful work. Moreover, the moving compute model exploits data locality better than the traditional atomic instruction based synchronization that suffers from shared data ping-pong between cores. These capabilities in the QUARQ architecture make it suitable for reuse oriented computing, where a single thread per tile maps to the underlying RISC-V based compute pipeline, and exploits the two-level cache hierarchy for efficient data access. To further exploit data reuse, QUARQ implements a tightly coupled memory (DTCM) that keeps the top of stack of the mapped thread local, thus avoiding accesses to the level-1 private cache. Figure 4 shows the baseline QUARQ architecture for the reuse mode of operation. To support throughput computing, QUARQ is extended with fine-grain multithreading capability in each tile with fast context switching. Depending on the needs of the algorithm

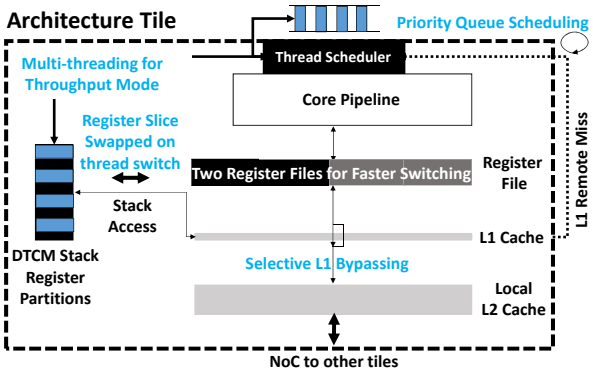

Figure 5: Proposed architecture for throughput mode of operation.

and input data, the number of threads per tile can be varied to tradeoff between reuse and throughput modes. The details of the throughput mode are outlined in in Figure 5, and described next.

Fine-grain Multithreading is implemented by supporting multiple contexts on the same tile (core). Contexts are worker threads, and swapped in and out of the the register file of the compute pipeline. Moreover, thread contexts require additional register space, which is acquired by using parts or all of the DTCM. However, in reuse mode, DTCM is fully utilized to keep top elements of the stack for the mapped thread. This space is reconfigured to map and store multiple thread contexts and a smaller portion of the stack in DTCM during the throughput mode of operation. Additional register contexts reduce space for the stack, showing a tradeoff with increasing number of threads. The size of the DTCM in the baseline QUARQ is restricted to 8 KB. Therefore, assuming a context size of 32 64-bit registers in RISC-V ISA, a maximum of 32 threads can be supported for storage in the DTCM. A switching hardware is assumed to support near-ideal latency for swapping threads between the DTCM and the register file of the compute pipeline. One option is to keep two register files, one to switch in the thread to be activated on the tile, while the other to hold the thread that is being deactivated and swapped out to DTCM. A fast one-way multiplexing logic is needed to swap a thread context from DTCM to the register file that is being activated. However, the movement of thread being swapped out can be hidden while the new thread performs its work. This paper assumes a single cycle thread swapping delay, and a microarchitecture implementation of such mechanism will be evaluated as future work.

Thread Switching policy is dependent on active thread's instruction count, cache misses, and synchronization related explicit messaging stalls. Unless a high latency event triggers a thread switch, a thread is switched by default after executing a fixed number of instructions (200 is used in this paper). A simple round-robin switching policy is implemented, in which the current thread is switched with thread+1, or the next thread ready for execution. Thread switching is also done on **L1 remote misses**, i.e., L1 cache misses that do not hit in the local L2 cache slice. These misses are served by a remote L2 cache slice or off-chip DRAM, all involving access through the on-chip network. This is similar to mechanisms used in GPUs, where many threads are virtually mapped onto a single core. The large number of threads make many data accesses, while thread switching hides the latency by overlapping with computations from other threads. The synchronization related explicit messaging stalls in the QUARQ's moving compute to data model also trigger thread switch since packets may take up to several tens of cycles before reaching their destination cores. This paper models a prototype thread switching mechanism, however other methods such as switching with priority are left as future work.

Selective Private L1 Cache Bypassing: The throughput mode of operation is expected to map many threads on a single tile, which effectively increases the private working set for the L1 cache. However, the L1 cache is assumed to be small since it needs to be low latency (single cycle access). Selectively bypassing the L1 cache has been explored to avoid cache pollution and thrashing [12]. Bypassing the L1 cache can be determined statically on the basis of application data structures, or dynamically based on the data type or program counter based heuristics. We envision the proposed architecture to exploit selective bypassing of the L1 caches, however this is left as future work and not evaluated in this paper.

3.1 Execution Model

The proposed architecture exposes reconfigurable reuse and throughput modes of operation, which must be selected when executing workloads. Multi-threading levels, L1 cache bypassing, and other architectural choices may be changed dynamically in this architecture. Once a particular benchmark-input combination is complete, the next combination is scheduled accordingly based on its characteristics. This model is expected to work well with streaming graph inputs or machine learning models. Our future work will explore prior work on HeteroMap [9] to build an accurate and fast performance predictor to select the right mode of operation in the proposed temporally reconfigurable architecture.

4. METHODOLOGY

The proposed architecture is implemented using an in–house industry–class simulator and the associated RISC-V toolchain. A futuristic tiled multicore processor with a two–level coherent private L1, shared L2 cache hierarchy per core, and on–chip interconnection X–Y routing–contention network models, is evaluated. Each single-issue core is mapped spatially to either a *service* or a *worker thread* based on the corresponding explicit messaging model. The default architectural parameters used for evaluation are shown in Table 1. Explicit messaging instructions use gcc extended asm blocks to direct the compiler to use specific registers. The models used here are ported from the Graphite multicore simulator [13]. In addition, the explicit messaging instructions, and the related protocol overheads are integrated into performance models. For workloads where cores are distributed into service and worker cores, multi-threading is only done on the worker cores, as preliminary results showed that multi-threading service cores does not improve performance.

4.1 Completion Time Metrics

Each benchmark is run to completion, and the completion time and energy consumption of *parallel* region is measured. The parallel completion time is broken down into the follow-

Architectural Parameter	Simulator
Number of Cores	256 RISC-V @ 1 GHz
Compute Pipeline per Core	In–Order, Single–Issue
Memory Subsystem	
L1–I Cache per core	16 KB, 4–way Assoc., 1 cycle
L1–D Cache per core	16 KB, 4–way Assoc., 1 cycle
L2 Inclusive Cache per core	64 KB, 8–way Assoc.
Directory Protocol	Invalidation–based MESI, ACKwise$_4$
DRAM Controllers	8 Contr., 10 GBps per Contr./ 100ns
Electrical 2–D Mesh with XY Routing	
Hop Latency	2 cycles (1–router, 1–link)
Contention Model	Only link contention, 64 bit Flits (Infinite input buffers)
Explicit Communication	
Send queue per core	16 words (4-entry)
Receive queue per core	64 words (16-entry)

Table 1: Architectural parameters for evaluation.

ing categories:
1) Compute Stall is the time spent retiring instructions, waiting for functional unit (ALU, FPU, Multiplier, etc.), and the stall time due to mis-predicted branch instructions.
2) Memory Stall is the stall time due to load/store queue capacity limits, fences, and waiting for load completion and L1 instruction cache misses.
3) Communication Stall is the stall time due to explicit messaging instructions. The *send* instruction is a non–blocking instruction that only stalls in case the tile's network router flow control temporally prevents injection of new messages into the network. The *sendr* is a blocking send instruction that requires an explicit reply. This is used in barrier synchronization, and offers a potential from throughput mode to exploit other threads to overlap with useful work. The *receive* instruction is a blocking instruction that stalls until a message is received in the service core's receive queue.

4.2 Benchmarks

Two graph workloads, Single Source Shortest Path (SSSP) and Triangle Counting (TC) are evaluated using the the California road network (CA) input graph from the SNAP repository [14]. An inception style machine learning workload, SqueezeNet [10] is also evaluated that classified an image from the ImageNet repository [15].

Real machine execution times of these workloads are also compared to the proposed architecture. The utilized machines are GTX-970 GPU, an Intel Core i7 multicore CPU, and an Intel Xeon Phi 7120P multicore. For GPU implementations, SSSP is picked from Pannotia [6], and TC is adopted from CRONO [16]. For Xeon Phi and Intel i7 implementations, both graph workloads are taken from CRONO [16]. SqueezeNet is acquired from Caffe [17], and an open source Intel implementation is used for results on Intel i7 machine [18]. However, the GPU implementation is adopted directly from Caffe [17].

5. PRELIMINARY RESULTS

To understand how throughput and reuse computing modes confer with workload scalability, each workload is evaluated on the proposed architecture by varying multithreading capability from a single thread to 64 threads per tile.

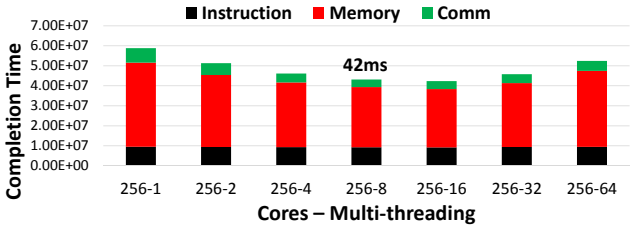

Figure 6: SSSP-CA scalability analysis.

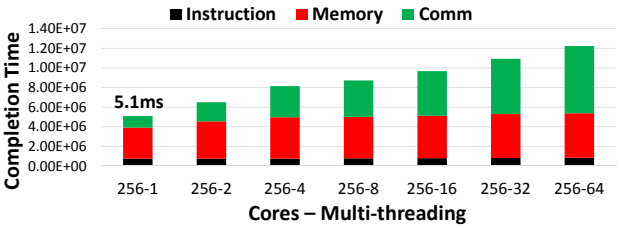

Figure 7: TC-CA scalability analysis.

SSSP: Figure 6 shows the completion times for the SSSP-CA workload with various multi-threading counts. For the evaluated 256 cores setup, the workload scales up to 8 threads per core. At a single thread per core, memory stalls dominate the completion time breakdown. As more threads are added per tile, the reduction in memory stalls happen since threads are switched to hides access latency with other useful work. In single thread mode, the workload exhibits thread work imbalance due to dissimilar work among the worker cores, as well as service and worker cores. With multithreading, the barrier synchronization switches threads due to communication stalls, and in turn creates more load balanced execution for worker cores. Therefore, with multithreading both memory stalls and communication stalls are seen to improve. SSSP-CA scales to 8 threads, which shows its throughput mode orientation. Compared to the GPU implementation from Pannotia [6], which takes 47 milliseconds, the proposed QUARQ's throughput mode takes 42 milliseconds. The evaluation on reuse oriented machine, Xeon Phi shows that SSSP takes 228 milliseconds. It is clear that SSSP benefits greatly from throughput oriented computing.

TC: We also take the case where the throughput mode does not help, and reuse mode is the right choice for best performance. Figure 7 shows the performance scaling breakdown for Triangle Counting (TC). It is seen that TC only benefits from reuse mode, and does not scale with additional threads. This happens because TC suffers from high inter-thread communication due to fine grain synchronization on shared data structures. This is best handled with the moving compute to data model in QUARQ and the underlying interactions of explicit messages and hardware cache coherence to exploit data access reuse with a single thread per tile. When more threads are added to each tile, the communication stalls rapidly increase, thereby degrading performance. In comparison to other machines, the QUARQ architecture achieves 5.1 milliseconds, compared to 67 milliseconds for the GTX-970 GPU, and 133 milliseconds for the Xeon Phi multicore. This show a more than 10× advantage using QUARQ's reuse computing mode.

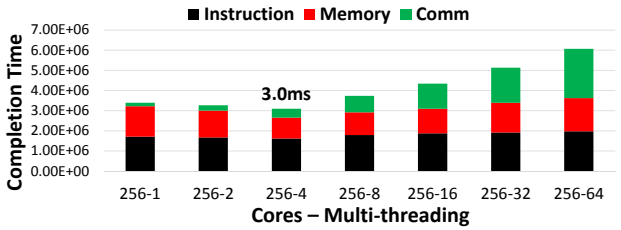

Figure 8: SqueezeNet scalability analysis.

SqueezeNet: As seen in Figure 8, SqueezeNet scales to 4 threads per tile. It is observed that compute is still a primary component in the completion time distribution, thus further scalability to higher core counts is expected for this workload. Although multithreading enables some reductions in memory stalls as thread counts are increased, the communication stalls start increasing as soon as more than one thread is utilized per tile. SqueezeNet is an optimized implementation that load balances work between barrier synchronization events. Therefore, switching threads with multithreading do not present much opportunities to hide communication stalls since all threads tend to reach barrier synchronization is close temporal proximity. However, additional threads now participate in barrier synchronization, and thus communication stalls increase with the increasing number of threads. For real machine comparisons, SqueezeNet takes 3.5 milliseconds on the Intel i7 multicore, and more than 2.5 milliseconds on the GPU. On the proposed architecture, SqueezeNet takes 3.0 milliseconds, which translates to $\sim$400 frames per second classification of images.

In summary, it is shown that the proposed QUARQ architecture can exploit workloads that require both *throughput* and *reuse* computing modes. The proposed architecture is also shown to outperform real-machine setups with equivalent or higher compute capabilities. Multithreading capabilities, along with the hardware cache coherence and moving compute to data synchronization models allow the extended QUARQ architecture to achieve higher performance in a situational setting. In future work, we plan to optimize more workloads from the domains of graph, machine learning and database processing, as well as architectural optimizations such as thread scheduling and placement strategies. We also plan to utilize and extend the HeteroMap's performance predictor for the proposed temporal multi-accelerator architecture. Selectively bypassing the private caches will also be explored to better utilize the available cache capacity for large number of threads in the throughput computing mode.

6. CONCLUSION

This paper shows that concurrency choices exist in graph and machine learning applications across various architectural capabilities. These choices primarily coincide with a workload being either throughout or reuse oriented, which is correlated with workload-input intrinsics. This work extends the QUARQ architecture that exploits reuse characteristics using hardware cache coherence and a novel moving compute to data synchronization model. The architecture is extended with reconfigurable multithreading capabilities to enable an efficient throughput computing mode. Results on three workloads show that the extended QUARQ architecture delivers best performance situationally. Some workloads only benefit from reuse computing mode, i.e., single thread per tile, while others benefit from a varying degree of multithreading per tile.

7. ACKNOWLEDGMENTS

This work was supported in part by Semiconductor Research Corporation (SRC). We would like to thank José A. Joao of ARM and Christopher Hughes of Intel for their reviews and feedback. We also thank Brian Kahne of NXP for his continued support on the QUARQ architecture.

8. REFERENCES

[1] S. Iqbal, Y. Liang, and H. Grahn, "Parmibench - an open-source benchmark for embedded multiprocessor systems," *Computer Architecture Letters*, vol. 9, no. 2, pp. 45–48, Feb 2010.

[2] N. Satish, C. Kim, J. Chhugani, H. Saito, R. Krishnaiyer, M. Smelyanskiy, M. Girkar, and P. Dubey, "Can traditional programming bridge the ninja performance gap for parallel computing applications?" in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ser. ISCA '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 440–451.

[3] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, Oct 2009, pp. 44–54.

[4] M. C. Jeffrey, S. Subramanian, C. Yan, J. Emer, and D. Sanchez, "Unlocking ordered parallelism with the swarm architecture," *IEEE Micro*, vol. 36, no. 3, pp. 105–117, May 2016.

[5] Y. H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 367–379.

[6] S. Che, B. Beckmann, S. Reinhardt, and K. Skadron, "Pannotia: Understanding irregular gpgpu graph applications," in *IEEE Int. Symph. on Workload Characterization (IISWC)*, Sept 2013.

[7] Q. Xu, H. Jeon, and M. Annavaram, "Graph processing on gpus: Where are the bottlenecks?" in *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, Oct 2014, pp. 140–149.

[8] M. Ahmad, C. J. Michael, and O. Khan, "Efficient situational scheduling of graph workloads on single-chip multicores and gpus," *IEEE Micro*, vol. 37, no. 1, pp. 30–40, Jan 2017.

[9] M. Ahmad and O. Khan, "Exploiting heterogeneous parallel accelerators to improve performance in graph analytics, src techcon," 2017.

[10] H. Dogan, F. Hijaz, M. Ahmad, B. Kahne, P. Wilson, and O. Khan, "Accelerating graph and machine learning workloads using a shared memory multicore architecture with auxiliary support for in-hardware explicit messaging," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2017, pp. 254–264.

[11] H. Dogan and O. Khan, "Quarq: A novel general purpose multicore architecture for cognitive computing, src techcon," 2017.

[12] G. Kurian, O. Khan, and S. Devadas, "The locality-aware adaptive cache coherence protocol," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 523–534.

[13] J. E. M. et. al., "Graphite: A distributed parallel simulator for multicores," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, Jan 2010, pp. 1–12.

[14] J. Leskovec and et. al., "SNAP Datasets: Stanford large network dataset collection," 2014.

[15] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, June 2009, pp. 248–255.

[16] M. Ahmad, F. Hijaz, Q. Shi, and O. Khan, "Crono : A benchmark suite for multithreaded graph algorithms executing on futuristic multicores," in *Proc. of IEEE Int. Symposium on Workload Characterization*, ser. IISWC, 2015.

[17] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, ser. MM '14. New York, NY, USA: ACM, 2014, pp. 675–678.

[18] OpenCV, "https://github.com/opencv/opencv/wiki/dnn-efficiency," 2017.