Journal of Systems Architecture 000 (2016) 1–11

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

A comprehensive reconfigurable computing approach to memory wall problem of large graph computation

Xu Wang[a], Yongxin Zhu[a,b,*], Linan Huang[a]

[a] School of Microelectronics, Shanghai Jiao Tong University, 800 Dongchuan Road, Shanghai 200240, PR China.
[b] School of Computing, National University of Singapore, Singapore.

A R T I C L E I N F O

Article history:
Received 1 October 2015
Revised 21 January 2016
Accepted 14 April 2016
Available online xxx

Keywords:
Embedded architecture
FPGA
Graph computing
Hardware implementation
Memory wall problem

A B S T R A C T

Graph computation problems that exhibit irregular memory access patterns are known to show poor performance on multiprocessor architectures. Although recent studies use FPGA technology to tackle the memory wall problem of graph computation by adopting a massively multi-threaded architecture, the performance is still far less than optimal memory performance due to the long memory access latency. In this paper, we propose a comprehensive reconfigurable computing approach to address the memory wall problem. First, we present an extended edge-streaming model with massive partitions to provide better load balance while taking advantage of the streaming bandwidth of external memory in processing large graphs. Second, we propose a two-level shuffle network architecture to significantly reduce the on-chip memory requirement while provide high processing throughput that matches the bandwidth of the external memory. Third, we introduce a compact storage design based on graph compression schemes and propose the corresponding encoding and decoding hardware to reduce the data volume transferred between the processing engines and external memory. We validate the effectiveness of the proposed architecture by implementing three frequently-used graph algorithms on ML605 board, showing an up to 3.85 $\times$ improvement in terms of performance to bandwidth ratio over previously published FPGA-based implementations.

1. Introduction

Graphs are widely used abstraction to express relationships between real-world data elements such as web graphs, telecommunication networks [1] and task [2], whose mining calculation can be abstracted into graph computing. Existing systems for executing graph computations are mainly targeted on general-purpose computers [3] or clusters of general-purpose computers [4–6]. However, they are usually inefficient at performing graph computations due to the irregularity of memory accesses causing large numbers of cache misses and the high data access to computation ratio causing stalls of the floating-point computation units.

The memory wall problem [7] is the key issue in graph computation. Although new memory devices such as Reduced Latency Dynamic RAMs (RLDRAM) are developed to address the problem, the compromise on memory capacity for access latency limits their usage on large-scale graph computation.

Reconfigurable computing based on Field Programmable Gate Array (FPGA) technologies becomes an attractive option to attack the memory wall problem of the graph computation for its availability of massive parallel on-chip resources with flexible interconnect to support fine-grained communication as well as abundant I/O pins to provide high off-chip memory bandwidth. Recent studies have leveraged such advantages to solve graph problems such as breadth-first search [8,9], all pairs shortest paths [10,11], and Sparse Matrix-Vector Multiplication kernels [12] in FPGA-based platforms. Many of them adopt a massively multi-threaded architecture that allows issuing multiple outstanding memory requests to the parallel memory banks of shared off-chip memory with dedicated hardware support such as the Convey HC-1 [13]. Although such architecture can exploit the memory controller bandwidth by keeping it busy with every clock cycle either for writing or reading, there is still a large gap from its peak memory access performance. Because the irregular memory access patterns in the graph-based algorithms can cause more page misses in DRAM memories, so the memory throughput is decreased due to the relatively long memory latencies.

In this paper, we address the memory wall problem by taking advantage of sequential streaming bandwidth of external DRAM

* Corresponding author.
E-mail addresses: wang2002xu@gmail.com (X. Wang), zhuyongxin@sjtu.edu.cn, yongxin.zhu@nus.edu.sg (Y. Zhu), huanglinan@sjtu.edu.cn (L. Huang).

http://dx.doi.org/10.1016/j.sysarc.2016.04.010
1383-7621/© 2016 Elsevier B.V. All rights reserved.

memory. Considering the fact that nature graphs have a much larger edge set than vertex set, access to edges and updates dominates the processing cost. Therefore, we adopt an edge-streaming model motivated by X-Stream [14], which iterates over edges and updates rather than over vertices. In our design, we stream edges from external DRAM memory while makes random access to the set of vertices in on-chip SRAM, leading to a fully utilization of external memory bandwidth in burst mode.

In order to support larger graphs which vertex data is not able to fit into the on-chip memory, we extend the edge-streaming model with massive partitions. In stead of shuffling the intermediate update results directly into its destination processing engine (PE), as we did in our previous work [15] to process small graphs, we wrote the intermediate updates back to slow storage (external DRAM memory), and read again when needed by the execution unit. We further analyzed several possible partition and workload assignment solutions to support such extension and proposed an optimized architecture with fine-grained partitions to achieve a more balanced load assignment among both PEs and on-chip memory banks.

Shuffling updates from all PEs into a much larger number of partitions leads to a significant challenge in designing the shuffle network architecture. In order to maintaining our design goal of exploiting sequential-access bandwidth to memory, it needs more on-chip memory to buffer shuffled updates. To address the problem, we proposed a two-level shuffle network, where each PE first shuffles the updates into several macro-partitions and then in the second level, these macro-partitions are further shuffled into final partitions by dedicated shuffle engines in parallel. The two-level shuffle network is able to reduce the on-chip memory requirement significantly while provide high processing throughput that matches the bandwidth of the external memory.

In addition to increase the utilization of the external memory bandwidth that improves the overall memory performance, we further ease the memory-bounded bottleneck by reducing the demand for memory access. We introduce a compact storage design based on graph compression schemes to reduce the data volume transferred between the processing engines and external memory. We also propose the hardware design of encoding and decoding logic to offload the computation overhead involved with graph compression while not break the streaming access pattern of external memory. With this compression scheme, data streams from the external memory can carry more information within the same memory bandwidth, which can in return increase the system throughput in terms of edges processed per second.

To verify our architecture, we experiment three graph algorithms under six different graphs in ML605 board. The major contributions of this work include:

- The extension of the edge-streaming model with massive partitions for reconfigurable hardware acceleration of graph problems, which supports larger graphs while provides a better load balance.
- A two-level shuffle network architecture to significantly reduce the on-chip memory requirement while provide high processing throughput that matches the bandwidth of the external memory.
- A compact storage design based on graph compression schemes and the corresponding encoding and decoding hardware to reduce the data volume transferred between the processing engines and external memory.
- Verification of our extended architecture on a ML605 system using three de-facto benchmark with detailed comparison of the state-of-the-art hardware implementation, showing up to 3.85 times improvement in terms of performance to bandwidth ratio.

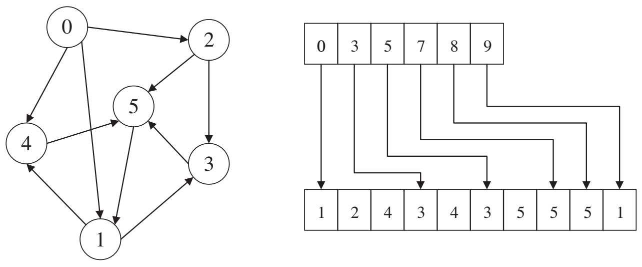

Fig. 1. Example of CSR representation.

The remainder of this paper is organized as follows. Section 2 introduces the background and motivation of our work. Section 3 presents our approach to address memory wall problem including the edge-streaming model with massive partitions, the two-level shuffle network architecture and the graph compression scheme. In Section 4, we verify our architecture with performance results. We review the related work in Section 5 and conclude our work in Section 6.

2. Background and motivation

2.1. Preliminaries

Generally, graph structure is abstracted as an ordered pair sets $G = (V, E)$, which comprises the set V of n vertices and the set E of m directed edges. An edge connecting vertex u and v can be denoted as $e = (u, v)$, where u is the source and v is the destination. Consequently, e is an outgoing edge and inbound edge of u and v respectively. For a given vertex u, if $\Gamma^+(u) = \{v \in V | (u, v) \in E\}$, then $\Gamma^+(u)$ is the set of all outgoing neighbors of vertex u and its out-degree denoted as $d^+(u) = |\Gamma^+(u)|$ is the number of edges with u as the source vertex. Accordingly, $\Gamma^-(u)$ and $d^-(u)$ are referred as the inbound neighbors and in-degree of u in a directed graph G, where $\Gamma^-(u) = \{v \in V | (v, u) \in E\}$ and $d^-(u) = |\Gamma^-(u)|$.

Arbitrary data can be associated with vertex u to retain state stored in u and the data is denoted as $\{D_u\colon u \in V\}$. Usually the value of D_u will be updated during the execution of graph algorithms while the topology of G remains unchanged.

2.2. Graph representation

Compressed sparse row (CSR) is a compact representation format of graph. The CSR format stores a whole graph into two arrays: vertex offset array $V[\,]$ and edge index array $E[\,]$. Edge index array contains destinations of outgoing edges of all vertices, which are sorted by their source vertex. Vertex offset array stores offsets of the first outgoing edge from each vertex. In CSR format, the out-degree of a given vertex u can be calculated as $d^+(u) = V[u + 1] - V[u]$. Correspondingly, compressed sparse column (CSC) format allows fast sequential access to the in-edges for vertices. These representations minimize memory use to $O(n + m)$, where n and m are the number of vertices and edges. Fig. 1 shows an example of CSR representation.

2.3. Memory wall problem of graph computation

Most of the graph algorithms can be expressed in vertex-centric model described in Algorithm 1. Each vertex collects and adds up the updates generated along its inbound edges by gather function $g(\cdot)$ and the sum will be used to modify vertex associated data field by apply function $a(\cdot)$. However, simply implementing the vertex-centric model does not enable efficient graph computation. Since for each vertex, all vertex value of its inbound neighbors are accessed by indirect memory reference under CSC format,