Database Query Acceleration with Reconfigurable Computing Fabrics

Cheng Liu

September 2, 2016

1 Motivation

FPGAs have been used as accelerators for high performance relation database applications. Compared to other parallel computing architectures such as multi-core processors and GPUs, FPGAs usually outperform on energy efficiency and can also achieve comparable performance in many cases. The use of FPGAs for database acceleration now has attracted increasing attentions of database researchers from both academia and industry.

Most of the previous work focused on building specific circuits for the typical compute-intensive operations involved in database query applications using either hardware description languages (HDLs) or high level synthesis (HLS) languages while leaving the hardware unchanged through the application lifetime. This explores the use of FPGAs for specific hardware accelerator design and will be good enough when the application doesn't change with time. However, there are many cases that the required FPGA accelerators may change with time.

For instances, the load of database query may change with time [2] [5] and different types of operations are needed accordingly. On top of the system fluctuation, the same operation such as sort operating on different attributes of the database which can be float, string, char or integer utilizes diverse FPGA resources. Unlike general computing architectures (i.e. CPU and GPUs) which typically use the same computing cores to handle all the different operations, FPGAs needs specific hardware for each types of operations. Therefore, some of the operations implemented may only be used for a short period of time while they may consume considerable hardware resources. In addition, some of the combination of the operations may be more efficient than the separate implementations [4] [1] and the combined implementations will be better when they are frequently used in the applications. But separate implementations will be preferred when the operations are not used at the same time. The mismatch between the database query and the underlying FPGA accelerators may result in severe performance and power efficiency degradation.

FPGAs which allow multiple granularities of reconfiguration using runtime mode selection, partial reconfiguration techniques or multiple FPGA contexts can be adapted to bridge the gap and provide hardware accelerators as needed or just enough for the sake of performance and power efficiency of database query

applications.

2 Main Idea

The basic database query processing system as shown in Fig. 1 follows the architecture that is used in omniDB [6] while the data-parallel primitives will be implemented on FPGA. From the perspective of omniDB, the FPGA is abstracted as a library providing all kinds of efficient data parallel primitives. Each primitive may have multiple instances implemented on the FPGA. All the primitives implemented on FPGAs are managed by a primitive resource manager. The manager will allocate a sequence of primitives to the high level data query operators. When there are abundant hardware primitives, the manager will allocate hardware operators as needed which essentially make best use of spacial resources providing the best performance. When there are only limited hardware primitives, the manager will provide just enough primitives to high level operators which try to temporally reuse the hardware resources.

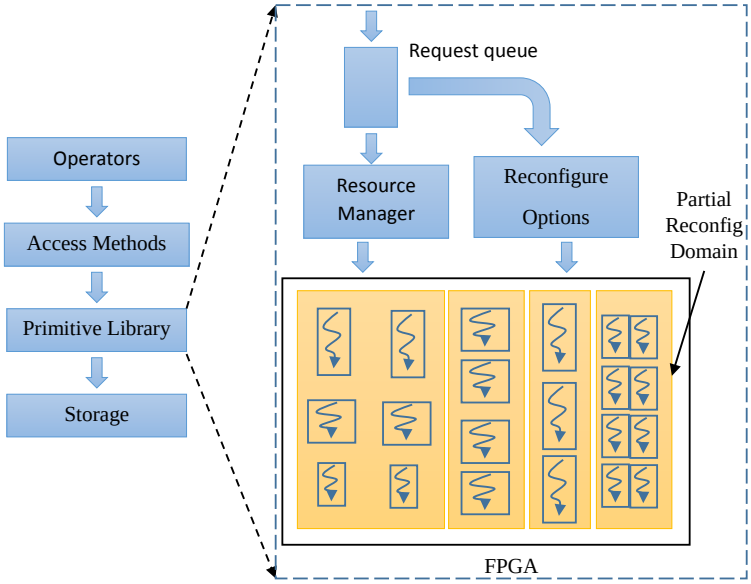

Figure 1: Database Processing Architecture Using FPGA Accelerators

To make best use of the hardware accelerators, a request queue will be maintained in the manager. With the status of the queue, the underlying hardware

primitives that will be needed can be analyzed at runtime. Given this runtime information, the FPGAs can be reconfigured to match the requirements at runtime using partial reconfigurations or some other dynamic reconfiguration techniques. With some other high-level data query information, it is possible to reduce the frequency of the runtime reconfiguration and thus are beneficial to the overall system.

2.1 How long does it take to reconfigure FPGA?

There are typically three methods for FPGA reconfiguration as listed in Table 1 [3]. FPGA design with mode selection can change its functionality by simply modifying some selection configuration bits. This mode achieves the fastest configuration but it usually requires the design to include all the functionality. Thus it typically results in large resource consumption. Multi-context bitstream method is relatively slow and needs larger on-chip storage to keep the bitstream. In addition, it may need to reset the whole FPGA board. Partial reconfiguration provides a nice trade-off between configuration time and on-chip storage requirement.

Reconfigure Method	mode selection	partial reconfiguration	multi-context bitstream
Reconfiguration Time	1-10us	1-100ms	0.5-10s
Reconfiguration Size	1-100B	0.1-1000KB	0.1-10MB

Table 1: Reconfiguration Time

2.2 Data query variation

The authors in the papers [2] [5] motivated the dynamic resource management or allocation for database query system. The analysis timescale is from seconds to minutes. I didn't see any report on fine-grained analysis. Maybe I chose the inappropriate key words and more investigation will be needed. If the database query varies in seconds, current FPGA reconfiguration capability should be enough for runtime optimization.

3 Summary

Taking advantage of the FPGA's reconfigurability, the FPGA accelerators can be dynamically reshaped to provide more hotspot acceleration fabrics based on the runtime requirements of data query, which are beneficial to both the system's performance and power efficiency.

References

[1] Ren Chen and Viktor K Prasanna. Accelerating equi-join on a cpu-fpga heterogeneous platform.

[2] Cong Guo and Martin Karsten. Towards adaptive resource allocation for database workloads.

[3] Kyprianos Papadimitriou, Apostolos Dollas, and Scott Hauck. Performance of partial reconfiguration in fpga systems: A survey and a cost model. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 4(4):36, 2011.

[4] Mohammad Sadoghi, Rija Javed, Naif Tarafdar, Harsh Singh, Rohan Palaniappan, and Hans-Arno Jacobsen. Multi-query stream processing on fpgas. In *2012 IEEE 28th International Conference on Data Engineering*, pages 1229–1232. IEEE, 2012.

[5] Gokul Soundararajan, Daniel Lupei, Saeed Ghanbari, Adrian Daniel Popescu, Jin Chen, and Cristiana Amza. Dynamic resource allocation for database servers running on virtual storage. In *FAST*, volume 9, pages 71–84, 2009.

[6] Shuhao Zhang, Jiong He, Bingsheng He, and Mian Lu. Omnidb: Towards portable and efficient query processing on parallel cpu/gpu architectures. *Proceedings of the VLDB Endowment*, 6(12):1374–1377, 2013.