FPGA Related Project

Designing of a compiler for generating Verilog from a high level script

Objective is to develop a complier that can generate a synthesizable Verilog code of an ultra-low latency and low power data flow processor to achieve mathematical calculations according to a predefined script written in a high level language defined only to fit for the purpose. The Verilog code to be synthesized must accept a continues stream of data sets having around 100 data fields per set and generate an output steam of data sets according to the script. Though the script does not change after the Verilog synthesis, the requirement of the script is to synthesise different types of data flow processors quickly without a verification effort. Rationally behind not using a general purpose processor is that one input data set must be processed within less than 2 microseconds at 250MHz irrespective of script within the scope. It’s also good to have some debugging tool, in case of mismatch in generated Verilog.

Example:

Input set : {‘A’, 12.45, 30, 3, 1, ‘x’, ‘y’, “ABC”}
Output set: {12450, 0, 55, ‘x’, 0x010203} {0}
Script:
Let’s take Input: {x1,x2,x3,x4,x5,x6,x7,x8} and Output: {y1,y2,y3,y4,y5}

```
If (x1 == ‘A’) {
        y1 = x2 * 1000

        if (x2 > 777.77){
                y2 = 1
        } else {
                y2 = 0
        }

        y3 = x3 * x4 / 2  +10

        if (x5 == 1) {
                y4 = x6
        }else{
                y4 = x7
      }

      Y5 = bytewise(x8 – 64)
} else {
        y1 = 0
}
```

GPU Related Projects

01.Self-grouping data structures based on the usage patterns for real-time GPU processing

Structure of arrays based data structures well perform with GPUs and many other HPC technologies from higher level of cache hits due to strided memory access. In a system where real-time computations are done based data updates, computations can be done for data located in specific memory address ranges if the data has been grouped using dimensions which gets frequently changed and based on the group sizes and other factors.

Based on the differences of the integrated external systems, nature of the deployment, expected characteristics can get changed invalidating the original assumptions of the system bringing the system's performance down. This project is to solve this problem by ordering (grouping data) based on characteristics of the receiving data updates. System will determine the grouping of data structures on system startup or during runtime based on the past data.

02.Self-accelerating processing workflows (GPU vs CPU with insights to optimize)

GPU acceleration for computations are best suited for high throughput, compute intensive problems. Practically, when it comes to real-time actual systems, there are also these edge cases which doesn't pair well with the criteria for GPU processing.

For example,

1. During off peak hours the less load in the system might do well in CPU than CPU. In a latency critical real-time system these can cause performance problems.
2. In a hypothetical problem, there can be a region of a country with less population which needs to do less calculations based on the mathematical model compared to the populated regions where less computation instances as best done in the CPU performance wise.

This problem is to build a framework which is self-learning and choose between CPU and GPU level processing based on the past experiences. The framework will also provide useful performance matrixes for further improvements. This is similar to branch prediction in CPU executions applied to CPU vs GPU executions.

03.Multi server GPU processing workflows using RDMA data transfers.

In a GPU accelerated distributed systems, processing can happen in multiple GPUs installed in multiple servers. Transferring data between GPUs of multiple servers will have a quite a noticeable latency in a traditional approach, where data is moved from GPU -> CPU_Mem -> CPU -> Network -> CPU -> CPU_Mem -> GPU. This also might involve memory serializations and deserializations as well as dynamic memory allocations and deallocations for thousands of individual items.

Latency of such a processing workflow can be massively reduced, if a defined memory segment can be moved to a remote GPU using a RDMA technology in a way that the GPU in the remote machine can consume the memory for its processing as it is.

NVIDIA GPU direct RDMA is a similar technology but this requires an Infiniband network interface by Mellanox which is expensive and are not available in most hardware setups including some cloud environments.

04.Efficient GPU memory transfers for Real-time system

Host and GPU memory space which can maintain complex data structures as structure of arrays were the data structures are not necessarily similar. In a real-time system, arbitrary updates to the Host(CPU) memory can happen which requires an update to the GPU memory prior to the next computation. Tracking the exact data point is not always efficient. Depending on the design, a full memory copy might be cheaper compared to finding and mapping the delta change.

Implementing a system which is cable of efficiently transferring the changed delta data set to the GPU which minimal latency. Some possibilities are to

1. Find out whether unified memory is an ideal candidate on 6.0 Pascal architecture with CUDA 10 which has hardware level page migration implemented during a page fault.
2. Find out if a columnar data structure which can efficiently filter the changes using similar capabilities available in SQL such as group by, sort by , filter etc.. to fetch data can be used

05.(OmniSci)Mapd extension for custom kernel implementations

OmniSci is an open source GPU accelerated database which provides amazing low latency query capabilities. OmniSci uses its own data structures, partitioning and caching techniques to execute SQL queries on the database. Currently the available functions are limited to SQL capabilities. This project proposes an extension to the OmniSci implementation which can be used to execute any GPU supported mathematical expression on the already loaded or paged memory structures resides in the GPU.

06.Suitability of Actor frameworks on GPUs for real-time multi server systems.

The actor model in computer science is a mathematical model of concurrent computation that treats "actor" as the universal primitive of concurrent computation. In response to a message it receives, an actor can: make local decisions, create more actors, send more messages, and determine how to respond to the next message received. Actors may modify their own private state, but can only affect each other indirectly through messaging (obviating lock-based synchronization) – Source “Wikipedia”.

GPU at the sometime provides massive concurrent compute capabilities which is likely to benefit the actor processing model. This project is to evaluate the feasibility and implementing an actor framework which natively uses GPUs for its computation.

07.Database interface for GPUs

As at now, GPU computations which requires data to be fetched from a database has to be fetched from Database (Example: using ODBC) to Host memory and to GPUs respectively prior to invoking the GPU compute operation.

This project proposes a standard and portable interface to load data from a database (example: Oracle, SQL server, AWS EBS etc.) to the GPU directly, possibly by passing the Host (GPU) pipeline.