Hardware Based Floating Point Processing

Michael Parker
Product Planning
Altera Corporation
San Jose, California
mparker@altera.com

Colman Cheung
System Engineering Group
Altera Corporation
San Diego, California
ccheung@altera.com

***Abstract*—High performance floating point processing is increasingly required for many commercial and government and military applications. Typically, this is implemented using software on specialized multi-core CPU architectures. New techniques now allow for high performance floating point to be implemented in hardware architecture, specifically FPGAs. As with fixed point processing, the FPGA based hardware approach can provide a dramatic increase in processing throughput, with reduced power consumption and deterministic latency.**

This paper will review devices and methods for achieving consistent high performance system implementations in floating point, with single device designs at over 200 GFLOPs with FPGAs at the 40nm process node, and 1 Teraflop at the latest 28nm process node. Several FPGA vendors have long offered floating point operator libraries such as multiply and add/subtract which have similar areas, performance levels, and latencies. The combination of multiple arithmetic operators into higher level functions such a vector dot product operator are inefficient, and suffer from significantly reduced F_{max}. Typical latencies for both multipliers and adders are in the range of 10; a dot product operator with a few tens of inputs may therefore exceed a latency of 100. Routing congestion and datapath latencies are have been critical restrictions on floating point implementation on FPGA architectures. Parallelism is a key advantage of a hardware solution like FPGAs, but it is often not applied to floating point signal processing because the long latencies make the data dependencies in algorithms such as matrix decomposition difficult to manage. To date, the resultant systems offered poor performance levels, uncompetitive to other platforms such as GPU or multi-core CPU.

There are several ways in which these FPGA challenges can be mitigated. The FPGA can have floating point precision supported in the embedded DSP Blocks. More efficient ways of mapping the floating point datapath to the relatively limited routing structures can be designed. Rather than building up a datapath from individual operators, the datapath can be considered as a single function, with inter-operator redundancy factored out. Mantissa representation can be converted to hardware friendly twos complement, and mantissa widths extended to reduce the frequency of normalizations. Elementary functions can be implemented as much as possible using hard multipliers, which offer guaranteed internal routing and timing, as well as low power and latency. New techniques can be applied for matrix decompositions, with the algorithms restructured to remove most of the data dependencies, so that parallel – and therefore high latency – datapaths can be used for computation

Furthermore, complex algorithms and performance will be described in detail. A representative application requiring high performance floating point processing is Space-Time Adaptive Processing (STAP) radar. STAP is an advanced signal processing technique that is increasingly employed in radar applications to suppress interference. It can provide an improvement in the detection of slow moving targets that are obscured by clutter or jamming, which makes it particularly suitable for airborne surveillance, where the search for slow moving targets in severe clutter is a common scenario.

This paper will detail FPGA implementation of the critical processing in STAP, namely the matrix inversion required to implement this adaptive algorithm at high throughput. For voltage domain STAP, the QRD algorithm is often used, due to the non-square nature of the complex data array to be processed. Implementation of a complex QRD processing of a [200x192] sample matrix will be described.

For power domain STAP, the Choleski algorithm is often used, due to the efficiency of processing square matrices making up the radar data cube. Here implementation of a complex QRD processing of a [256x256] sample matrix will be used.

In both cases, both the decompositions and forward/back substitution will be detailed. By showcasing these complicated algorithms, the nature of the toolflow, design entry and native support for vector processing can be easily seen, providing a reference point for design of any complex algorithm requiring the dynamic range and precision of floating point numerical representation.

The paper will also discuss how such algorithms are structured to take advantage of the native parallelism of the FPGA, and to avoid data dependencies and stalling the major processing blocks. The trade-offs between the chosen vector size and the influence on processing throughput and device resource usage are discussed. Multiple benchmarks will be detailed on both 40nm and 28nm FPGAs, providing a range of throughput from 1000 matrices per second to well over 10,000 matrices per second within a single device.

The paper will conclude with discussion and analysis on the verification of the results, both in relating the integrity of FPGA results back to simulation model using common test vectors, as well as a comparison of the precision of the hardware based floating processing to that of a standard method, such as using Matlab on a Pentium CPU. It will be shown that the FPGA accuracy results are statistically superior to that of conventional IEEE754 representation using a processor architecture.