Implementation of Floating-Point VSIPL Functions on FPGA-Based Reconfigurable Computers Using High-Level Languages

Malachy Devlin[1], Robin Bruce[2] and Stephen Marshall[3]
[1]Nallatech, Boolean House, 1 Napier Park, Glasgow, UK, G68 0BH, m.devlin@nallatech.com
[2]Institute of System Level Integration, Alba Centre, Alba Campus, Livingston, EH54 7EG, UK, rbruce@sli-institute.ac.uk
[3]Strathclyde University, 204 George Street, Glasgow G1 1XW, s.marshall@eee.strath.ac.uk

Introduction

The evolution of FPGAs have advanced to the stage where high-capacity devices offering dedicated silicon for accelerating mathematical operations are available. The latest families offer up to 512 dedicated MAC (Multiply and Accumulate) units providing a performance capability of 256GMACs/s on 18-bit data. Complemented by a programmable-logic fabric, this allows for algorithm-dependent implementations. Typically, when FPGAs have been used to achieve high-performance computing solutions it has been with integer and bit-level algorithm implementations only. There has been a perception that microprocessors are still required to implement floating-point algorithms and that FPGAs are weak in this area. This is no longer the case, and FPGAs are now capable of implementing effectively both single and double-precision floating-point algorithms.

Floating-Point Cores on FPGAs

FPGAs can now outperform microprocessors in carrying out both single-precision and double-precision floating-point operations.[1] Xilinx's V-II Pro is capable of offering over 25GFLOPS/s of single-precision peak performance, with approximately a quarter of this performance possible for double-precision floating point. The current trends in FPGA performance produced by advances in semiconductor technology will yield FPGA devices with a factor of three to eight more peak floating-point performance than comparable microprocessors by 2009.[2]

VSIPL

The Vector, Signal and Image Processing Library is an open standard API for highly efficient and portable computational middleware for signal and image-processing applications. This library has typically been targeted at microprocessors and is provided as a C or C++ API. This library is aimed at high-performance applications and therefore has capabilities of handling a range of data types, including integer and floating point.[3]

This paper will look at the implementation of these libraries on FPGAs, aiming to provide to the FPGA developer the same level of abstraction on the FPGA from this middleware functionality as the software developer has traditionally had with the original APIs on microprocessors.

[1]This research is sponsored by Nallatech in conjunction with the Institute for System Level Integration and the University of Strathclyde.

Tools and Hardware

All FPGA implementations are targeted to Nallatech reconfigurable computing systems based on commercial off-the-shelf products. This enables deployment in a range of form factors, such as VME, cPCI, PCI 104 etc, on which VSIPL can be targeted.

The communications between host and system, between functions and between multiple FPGAs takes place over DIMETalk networks, created using the DIMETalk design GUI application. Entities in the system are nodes in a packet-switched network, with routers to connect nodes and bridges to connect FPGAs[4].

The initial implementations will use Nallatech's single precision floating-point cores, with easy extension to their double precision versions. These are implemented specifically for Xilinx FPGAs and are optimized for resource and performance.[5]

Implementation of Floating-Point Algorithms on FPGAs

The work that this paper represents can be seen as laying the foundations in the development of a VSIPL-compliant API to allow application developers to take advantage of the capabilities of FPGA computing with no more expertise than they would require using the same API in conjunction with a standard microprocessor.

The first step towards this goal is to demonstrate the implementation of key floating-point VSIPL functions on FPGAs. These algorithms are taken from the TASP VSIPL Core Plus Plus profile of the VSIPL API v1.1. These algorithms are initially implemented as ANSI C functions. To implement these functions in hardware they must first be prepared for compilation to VHDL by re-structuring the code so as to aid a compiler in finding parallelism and opportunities for pipelining that will lead to time gains over serial execution.

Ideally, one would implement all desired VSIPL API functions at least once on a multi-FPGA system with associated data storage capabilities. This, however, is overly demanding on FPGA resource and is not a practical solution.

The next best solution would be to implement the system seen in figure 1 below. In this set-up each individual function is instantiated dynamically on the FPGA as and when it is required, according to the algorithm running on the host. This would require functions to be pre-compiled for hardware and a versatile communications network to handle the changing functions on the FPGA. To be a worthwhile contender to a microprocessor-only implementation the following relationship needs to be satisfied:

$$t_{total} = \sum_{N=0}^{N_f} \left[t_{boot} + t_{load}(N) + t_{exec}(N) + t_{unload}(N) \right] < t_{proc} \quad (1)$$

Where t_{total} is the total execution time for the entire algorithm, which consists of N_f VSIPL functions, in hardware, t_{boot} is the time taken to load the next function in the algorithm onto the FPGA, t_{load} is the time taken to provide the function with the necessary data, t_{exec} is the time taken to obtain a result and t_{unload} is the time taken to send the output data to its destination. t_{proc} is the processing time for the algorithm when implemented solely on a microprocessor.

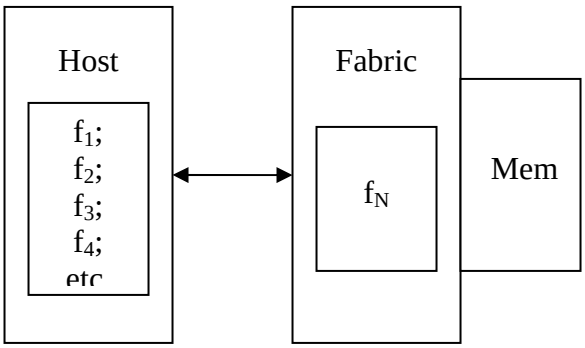

Figure 1: Ideal System Set-up

Before being able to implement the system shown in figure 1, which one could easily envisage as being VSIPL-compliant, one must overcome a number of difficulties. These range from problems with refreshing DRAM, bottlenecks in loading functions onto the FPGA and the potential limitations on communications latency and bandwidth.

The architecture shown in figure 2 below shows an initial set-up to demonstrate the performance of the VSIPL floating-point functions implemented in hardware. The entire algorithm is implemented in hardware with the host simply initializing the system, sending the input data and receiving the end result.

In order to achieve an improvement over a microprocessor-based implementation the following worst case relationship must be satisfied:

$$t_{total} = t_{boot} + t_{load} + t_{unload} + \sum_{N=0}^{N_f} \left[t_{exec}(N) \right] < t_{proc} \quad (2)$$

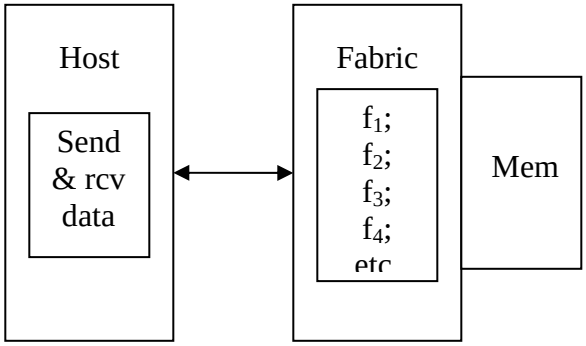

Figure 2: Initial System Set-up

The more functions an algorithm utilizes, the more silicon real-estate is taken up. Eventually on large algorithms a single device will no longer suffice for implementation. We will also therefore be demonstrating the partitioning and system control on multi-FPGA systems of these high-capacity algorithms.

References

[1] Underwood K, Hemmert K, *Closing the gap: CPU and FPGA trends in sustainable floating-point BLAS performance*, Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on (2004), pp. 219-228.

[2] K. D. Underwood. FPGAs vs. CPUs: Trends in peak floating-point performance. In Proceedings of the ACM International *Symposium on Field Programmable Gate Arrays*, Monterrey, CA, February 2004.

[3] *Vector Signal Image Processing Library*, VSIPL website, www.vsipl.org

[4] Craig Sanderson, *Simplify FPGA Application Design with DIMEtalk*, Xcell journal, winter 2004

[5] *Floating Point Math IP*, www.nallatech.com