2011 20th IEEE Symposium on Computer Arithmetic

Accelerating large-scale HPC Applications using FPGAs

Rob Dimond, Sébastien Racanière, Oliver Pell
Maxeler Technologies
rob@maxeler.com, sebastien@maxeler.com, oliver@maxeler.com

***Abstract*—Field Programmable Gate Arrays (FPGAs) are conventionally considered as 'glue-logic'. However, modern FPGAs are extremely competitive compared to state-of-the-art CPUs for commercial HPC workloads, such as those found in Oil and Gas and Finance. For example, an FPGA accelerated system can be 31-37 times faster than an equivalently sized conventional machine, and consume 1/39 of the power.**

The key to achieving the best performance in FPGA accelerators, while maintaining correctness, is optimization of arithmetic units and data types to suit the range/precision at each point in the computation. The flexibility of the FPGA to implement non-standard arithmetic, combined with a data-flow programming model that instantiates a separate unit for each arithmetic operator in the code provides a wide design space.

As such, FPGA computing offers significant opportunity for arithmetic research into 'large scale' HPC applications, where there is an opportunity to move away from standard IEEE formats, either to improve precision compared to the CPU version or to increase speed.

***Keywords*-FPGA; Acceleration**

I. Introduction

FPGAs represent one end of a spectrum of compute devices, with many thousands of very basic compute units interconnected on a chip. Conventionally, the FPGA is a logic emulation device used in telecommunications infrastructure and 'embedded' compute applications. However, advances in FPGA technology have given us devices that can implement HPC applications that would ordinarily run on microprocessors. The run-time configurability of the FPGA enables a custom computer with arithmetic and datapaths tailored for just one application, with finely tuned optimization at the application, arithmetic and bit level.

The limit to FPGA performance is either the depth and number of parallel pipelines on a single device (compute bound), or the data bandwidth between the FPGA and external memory (memory bound). In the compute bound case, we optimize the arithmetic to fit in as small an area as possible. In the memory bound case, we optimize the data representation to minimize the data that must be transferred at each compute step. The data-flow computation methodology allows us to individually optimize each and every point where data is computed and transferred[4].

In this article we present four case studies of commercial applications that achieve speedup and operational cost savings of several orders of magnitude using FPGA acceleration. In particular, we focus on the particular optimizations that make the large speedups possible.

II. FPGA Programming Model

We use a programming model based on data-flow which is entirely driven by Java. The user specifies *kernels*, which are statically scheduled, pipelined data-paths, and a *manager* that controls the routing of streaming data between multiple kernels and off-chip connections. Kernels use arbitrary precision floating/fixed point types and our compiler takes care of type conversions.

To evaluate precision and correctness, we compile data-path descriptions to C, for quick turnaround of simulation results. Testing larger data-sets — that would be too slow to simulate on the CPU — is possible with a 'single click' automated compile to the FPGA, with all the infrastructure for getting test data to/from the chip automatically instantiated.

III. Finite Difference Modeling

Modeling of wave propagation by 3D finite difference is an important application in geoscience [1] that runs on conventional CPU clusters with many thousands of nodes, for the purpose of locating oil and gas reserves beneath the earth's surface. Finite difference is a well known method for solving PDEs (the acoustic wave equation in this case).

The computation involves iteratively advancing in time a discretised wavefield (u in the equation below), using the acoustic velocity at each point (v below).

$$\frac{\partial^2 u}{\partial t^2} = v^2 \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \right) \tag{1}$$

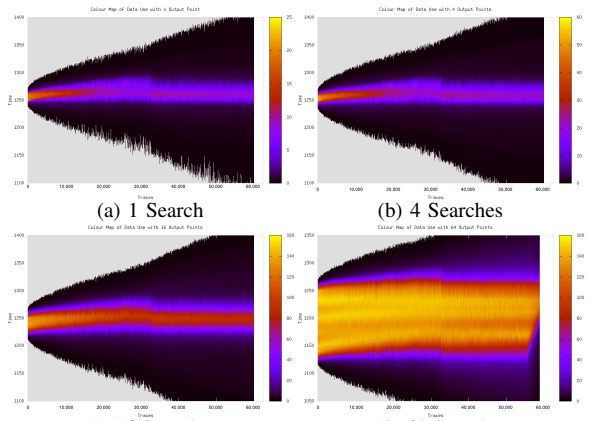
The discretised grid is typically 1000^3 in size and stored in a floating point representation. The core is a 3D convolution with a Taylor expansion of the above equation which occurs in either fixed or floating point arithmetic. The convolution operator is often separable so we have the option of implementing it directly, or taking multiple passes over the data with smaller operators. The key to performance on the FPGA is trading off the operator size — resource utilization of the FPGA — against the DRAM bandwidth required for multiple passes over the data.

The actual speedup of modeling varies with the convolution operator and data-set size, although two orders of magnitude over a single CPU core is typical.

IV. CRS Trace Stacking

CRS trace stacking is a geoscience application used to process data from seismic surveys. At the core of stacking is

1063-6889/11 $26.00 © 2011 IEEE
DOI 10.1109/ARITH.2011.34



(a) 1 Search (b) 4 Searches

(c) 16 Searches (d) 64 Searches

Figure 1: Usage (lighter = greater reuse) of an input datum in CRS optimization, when computing 1, 4, 16 and 64 t0 output points.

an optimization to maximize the Semblance function within an 8 dimensional parameter space. The application performs the parameter search repeatedly, for each and every output data point using a 4GB working set of complex, floating point numbers. For a CPU implementation, we compute all output data points serially. For the FPGA implementation, we compute many data-points in parallel.

The key to performance in stacking on the FPGA is to maximise reuse of each point loaded from DRAM, otherwise the available DRAM bandwidth into the chip is the limit to performance. Figure 1 shows the degree of reuse of each input data point while varying the number of output data points computed in parallel. With 64 parallel points the data reuse is high enough so that we are no longer memory bound.

The final FPGA implementation of stacking gives a speedup of more than 200x for a Maxeler MAX2 FPGA card compared to a single CPU core[2].

V. Credit Derivatives Valuation and Risk

Credit derivatives enable the transfer of default risk (a debtor failing to pay back a loan) in exchange for regular payments. Currently, financial organisations maintain large portfolios of these instruments and dedicate thousands of CPU cores to calculate value and risk daily.

The core of the application is a convolution that accumulates a distribution of loss — the probability of default on 0 to 100% of the value — from the individual probabilities of default on components of the financial instrument. Key to performance is unrolling the convolution in two dimensions (as shown in Figure 2) so that many arithmetic operations occur in parallel.

We use the flexibility of FPGA arithmetic to trade off precision and performance in different versions of the application. A full precision version, used for pricing accurate to 10^{-8} gives a 31x speedup over an 8-core Xeon E5430 server.

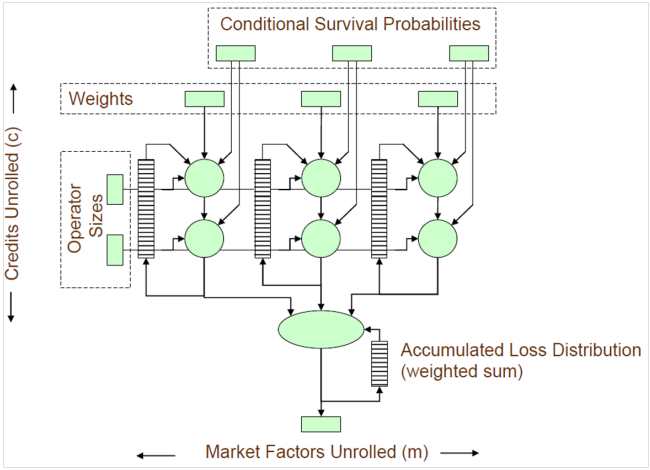


Figure 2: The convolution of survival probabilities to accumulate the loss distribution is unrolled in two dimensions. Circles represent arithmetic units (multiply adds) and rectangles are memory/delay elements.

A reduced precision version accurate to 10^{-4} gives a 37x speedup (270x per core) for rapid evaluation of scenarios. An interesting effect of acceleration is that the node containing the FPGA accelerator consumes less power (238W) than the same node running the application on the CPU (246W) [3].

VI. Conclusion

Computing with FPGAs provides significant advantages for several high-value cluster applications. Getting the best possible performance on FPGAs requires different tuning than for code running on CPUs. FPGA tuning typically involves customised arithmetic types for optimal resource utilisation, and/or storage types for optimal bandwidth. The large scale of parallelism also requires careful thought about reuse of data on the chip.

References

[1] O.Lintjorn, R.G.Clapp, O.Pell, O.Mencer, M.J.Flynn and H. Fu, *Beyond Traditional Microprocessors for Geoscience High-Performance Computing Applications* in IEEE Micro, Vol.31 Iss. pp.41-49, March-April 2011

[2] P.Marchetti, D.Oriato, O.Pell, A.M.Cristini and D.Theis, *Fast 3D ZO CRS Stack - An FPGA Implementation of an Optimization Based on the Simultaneous Estimate of Eight Parameters* in Proc. 72nd European Association of Geoscientists and Engineers (EAGE), Barcelona, June 2010

[3] S. Weston, J. Spooner, S. Racanière, O. Mencer, *Rapid Computation of Value and Risk for Derivatives Portfolios* to appear in Concurrency and Computation: Practice and Experience.

[4] H.Fu, W.Osborne, B.Clapp, O.Pell, *Accelerating Seismic Computations on FPGAs from the Perspective of Number Representations* in Proc. 70th European Association of Geoscientists and Engineers (EAGE), Rome, June 2008