Effective Floating Point Applications on FPGAs: Examples from Molecular Modeling*

Bharat Sukhwani Matt Chiu Md. Ashfaq Khan Martin C. Herbordt

Computer Architecture and Automated Design Laboratory
Department of Electrical and Computer Engineering
Boston University; Boston, MA 02215

ABSTRACT

While FPGAs have only one fifth the raw floating point capability of GPUs, other attributes allow them to be surprisingly competitive with respect to a number of critical floating point applications. In the first part of this extended abstract we review these FPGA attributes. In the second part we sketch three applications in molecular modeling where we have found FPGA performance to compare favorably: molecular dynamics, molecular docking, and discrete event molecular dynamics.

1. INTRODUCTION

With GPUs offering nearly one TFLOP peak performance, the competitiveness of FPGAs in floating point intensive applications (FP), which has always been tenuous, now seems improbable. Here we show that this is not necessarily so – rather, that the FPGA's flexibility, together with algorithmic restructuring, can yield competitive performance for some of the most demanding and critical FP applications.

We begin with some general points about FPGA floating point capability, beginning with peak numbers.

1. **Raw FP capability from IP cores.** Because FPGAs are flexible and because different FP cores have different resource requirements this number varies. For single precision on the Altera Stratix III SL340, we fit 144 multipliers and 215 adders, while leaving much of the routing logic still available. Running at 200MHz and assuming 100% utilization gives a peak performance of 72 GFlops. Other reports indicate up to 190 GFlops for the Virtex 5 SX240 (from www.xilinx.com).
2. **FP cabability with pipeline optimization.** In a pipeline of FP units, data are repeatedly unpacked and repacked. Removing these and other redundant operations, plus performing certain other optimizations, can improve the raw FP capability by more than 50% [3].
3. **Arbitrary precision.** If lower precision or range are acceptable, then non-standard representations of FP or fixed point can be used with a proportional increase in performance.

The true benefit of using FPGAs, however, lies in the fraction of raw performance that can be obtained.

1. **Overall.** FPGAs allow the construction of microarchitectures specific to a particular application. These generally include some number of dedicated FP pipelines.
2. **Feeding the pipelines.** FPGAs have collective BRAM bandwidth of 4TB/s. Moveover, this bandwidth can be directed to the pipelines with substantial flexibility.
3. **Communication among threads.** Communication among threads can often be designed to have nearly arbitrarily low latency.
4. **Synchronization** can be similarly designed.

As a result, it is common to design FP applications where, in the steady state, payload is delivered by every pipeline on every cycle. Since communication can often be managed in the background, even with set-up and tear-down, 50% utilization or more can be achieved (see, e.g., [1]).

On GPUs and CPUs, achieving such high utilizations is difficult. As an example, we look at a high-impact and highly tuned application: the FFT. On the NVIDIA Tesla C1060 we run a 128^3 FFT in 6.3ms for a utilization of less than 4% (using the NVIDIA CUFFT library and not counting transfer time). On a 2GHz Intel Xeon quad-core CPU the same FFT runs in 58ms for a utilization of about 16% (using FFTW www.fftw.org and assuming a peak of 24GFlops). Please note that these results represent a non-scientific sample, there are certainly applications for which much higher utilizations can be achieved.

From this discussion it follows (trivially) that FPGAs can get competitive performance on FP applications when there exists a good mapping and especially when single precision or less is tolerable. For the rest of this abstract we discuss three such applications: molecular dynamics (MD), molecular docking (docking), and molecular dynamics based on discrete event simulation (DMD).

2. MOLECULAR DYNAMICS

MD simulation is an iterative application of Newtonian mechanics to ensembles of atoms and molecules. The bulk of the computation is in calculating the short-range force between all particle pairs i and j:

$$\frac{\mathbf{F}_{ji}^{short}}{\mathbf{r_{ji}}} = A_{ab} r_{ji}^{-14} + B_{ab} r_{ji}^{-8} + QQ_{ab}(r_{ji}^{-3} + \frac{g'_a(r)}{r}) \quad (1)$$

where A_{ab}, B_{ab}, and QQ_{ab} are distance-independent coefficient look-up tables indexed with atom types a and b.

The key aspect of MD with respect to this discussion is that the complexity of each the computation $\mathbf{F}_{ji}^{short}$ depends on the r_{ji}. When $r_{ji} > r_c$ then $\mathbf{F}_{ji}^{short} = 0$ and no force needs to be computed at all. On the other hand, when r_{ji} approaches the van der Waals radius r_{vdW}, then the r_{ji}^{-14} term dominates and maximal precision is needed. For most of $r_{vdW} + \epsilon < r_{ji} < r_c$ single precision is sufficient for most of the computation.

The overall design is shown if Figure 1. The main idea is to process the bulk of the particle pairs quickly and with little hardware but to allocate sufficient hardware for the particle pairs that need it. Processing proceeds as follows.

1. The first step takes place on the host: sorting particles into cells. This operation takes only a few hundred microseconds.
2. Processing on the FPGA proceeds cell-by-cell. For each "home" cell, the cell neighborhood is loaded from off-chip memory (POS SRAM into POS Cache).

*This work was supported in part by the NIH through award #R01-RR023168-01A1. Web: www.bu.edu/caadlab. Email: {herbordt|bharats}@bu.edu

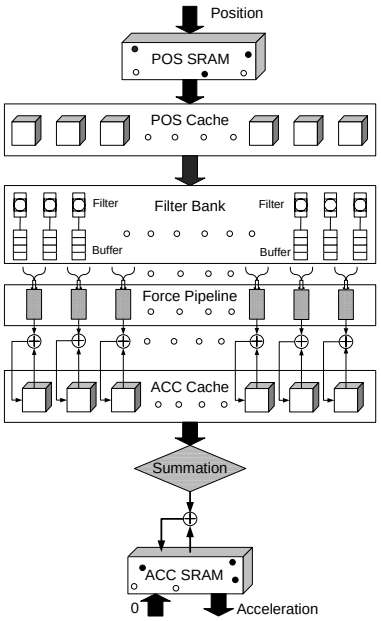

Figure 1: Schematic of the HPRC MD system.

3. Now for each particle i in the home cell and for each particle j in the cell neighborhood, r_{ji} is computed (Filter Bank). Particle pairs with $r_{vdW} + \epsilon < r_{ji} < r_c$ are passed on to the Force Pipelines.
4. The forces between the remaining particle pairs are computed and accumulated.

Steps 1 and 2 construct the cell lists, Step 3 the neighbor lists, and Step 4 the actual force computation.

In our current system (see [1]), based on a Stratix-III SE260, there are 10 force pipelines and 8 filter pipelines per force pipeline. For operating frequency we currently achieve 200MHz. The design operates at over 90% efficiency. For the 90K particle ApoA1 benchmark, the short-range force for a single iteration is computed in less than 20ms for a 90-fold per-core speed-up.

3. MOLECULAR DOCKING

Molecular docking refers to the non-covalent bonding between molecules. Docking is often computed by mapping the molecules' characteristics to 3D grids. The most energetically favorable relative position is then found by summing the voxel-voxel interaction values for each modeled force at all positions, to generate a score, and then repeating this for all possible translations and rotations. On serial computers (and GPUs) this computation is cast as a series of FFTs.

There are two key aspects of the docking computation for this discussion. The first is that the molecules' grid characteristics are low precision with 7-8 bits generally being sufficient. The second is that FPGAs perform direct convolution at very high efficiency (see [4]).

As a result for small molecule docking (one molecule 128^3, the other up to 12^3) the FPGA achieves a per-core speed-up of 25x to 35x. For protein-protein docking (both molecules up to 128^3) the superior asymptotic complexity of the FFT takes over and the GPU is superior [5].

4. DISCRETE MOLECULAR DYNAMICS

In contrast with traditional MD, molecular dynamics simulation based on discrete event simulation (DMD) uses simplified discretized models enabling simulations to be advanced by event rather than by time-step. A DMD system consists of the

- **System State**, which contains the particle characteristics: velocity, position, time of last update, and type;
- **Event Predictor**, which transforms the particle characteristics into pairwise interactions (events);
- **Event Processor**, which turns the events back into particle characteristics; and
- **Event Priority Queue**, which holds events waiting to be processed ordered by time-stamp.

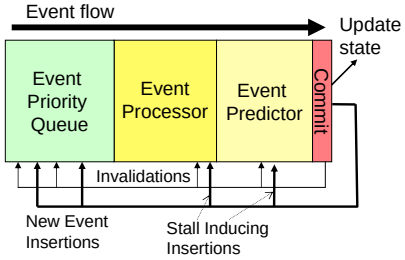

Figure 2: DMD with a pipelined event processor.

The main idea in the FPGA design is to process DMD in a single pipeline (as shown in Figure 2). That is, while a large number of events can be processed simultaneously, at most one event at a time is *committed*. Viewed another way, this design is of a microarchitecture that processes events rather than instructions: the logic is analogous to that used in modern high-end CPUs for speculative instruction execution. And as with a high-end CPU, keeping the pipeline moving requires complex high-bandwidth low-latency communication.

Current DMD codes are highly efficient and process events in about 10us. Since execution is chaotic, parallelizing DMD is challenging. Event executions can cause events to be invalidated and inserted anywhere in the queue. With substantial care (e.g., hand-written locks) we achieve speed-ups with four cores of a multicore CPU [2]. On the FPGA we execute events at a small multiple of the clock rate for a speed-up of 50x-100x.

5. CONCLUSION

We have described three floating point intensive applications where FPGAs are highly competitive. In each case the application lends itself to the strengths of the FPGA: flexible precision; support for communication that is complex, lowlatency, and high bandwidth; and support for custom FP pipelines.

6. REFERENCES

[1] Chiu, M., and Herbordt, M. Efficient filtering for molecular dynamics simulations. In *Proc. IEEE Conference on Field Programmable Logic and Applications* (2009).

[2] Herbordt, M., Khan, M., and Dean, T. Parallel discrete event simulation of molecular dynamics through event-based decomposition. In *Proc. International Conference on Application Specific Systems, Architectures, and Processors* (2009), p. TBD.

[3] Langhammer, M. Floating point datapath synthesis for FPGAs. In *Proc. IEEE Conference on Field Programmable Logic and Applications* (2008), pp. 355–360.

[4] Sukhwani, B., and Herbordt, M. Acceleration of a production rigid molecule docking code. In *Proc. IEEE Conference on Field Programmable Logic and Applications* (2008), pp. 341–346.

[5] Sukhwani, B., and Herbordt, M. GPU acceleration of a production molecular docking code. In *Proc. General Purpose Computation Using GPUs* (2009).