Obtaining High Performance via Lower-Precision FPGA Floating Point Units

Junqing Sun
Advisor: Gregory D. Peterson
University of Tennessee, Knoxville
[jsun5, gdp]@utk.edu

1 PROBLEM & MOTIVATION

Higher-precision floating point data formats are widely used because of their high accuracy and wide data range. However, their significant resource cost and slow speed are prohibitive for many timing-critical applications. For example, a double-precision multiplier requires four times the hardware of a single-precision multiplier and requires much longer computation time [1]. To avoid expensive and slower double precision floating point units, many applications use lower-precision floating point or even fixed point data formats to achieve higher performance by sacrificing accuracy. Unfortunately, these algorithms might fail to converge or may produce unexpected results if their data formats do not have sufficiently high accuracy. Moreover, finding the simplest data formats required by real applications usually involves difficult analysis [16].

Unlike previous work, our research aims at a much more aggressive goal - achieving higher performance without losing any of the accuracy of higher-precision data formats by using mixed-precision algorithms and architectures. This goal has been met with both mathematical analysis and experimental results [1]. In this paper, we first analyze the performance of different floating point data formats. Second, we introduce the mixed-precision algorithm and architecture. Our supercomputer implementation results show that a mixed-precision algorithm and architecture can significantly improve performance without losing accuracy. Based on our mathematical analysis and experimental results, we propose mixed-precision architectures for future high performance reconfigurable computers.

1.1 Floating Point Performance on FPGAs

Field programmable gate arrays (FPGAs) have shown great potential in accelerating computationally intensive applications because of their intrinsic parallelism, pipelining, and flexible architecture. Previous work concluded that peak floating-point performance on FPGAs exceeds that of CPUs and will increase by orders of magnitude [6]. Our previous work indicates the cost and performance of floating point IP cores directly determines the performance of many computational intensive applications [7]. Because of the importance of floating-point data formats, many researchers and commercial vendors have developed floating-point intellectual property (IP) cores for FPGAs. Xilinx, a large FPGA vendor, has included configurable pipelined floating-point operators in its development tools (ISE) [11]. Related floating-point IP cores have also been developed by academic research groups [12][13]. To reduce hardware resource cost and achieve high performance, operators can be built from either combinational logic slices or embedded circuits such as built-in 18x18 multipliers. For commonly used floating point components, the resource cost increases *linearly* for adders and *quadratically* for multipliers. We estimate the floating point performance of FPGAs by multiplying the frequency and maximum number of IP cores that fit on the FPGA. Figure 1 shows the floating point performance of a Xilinx Virtex 4 XC4LX160-10 FPGA. For this test, we assume 70% of the slices can be configured as floating point units, while the rest are available for routing and control. Our test results reveal the floating point performance for lower-precision data formats significantly exceeds that of higher-precision data formats. The customized data format s16e7 (16 bit mantissa and 7 bit exponent) outperforms double precision (s52e11) by a factor of 23x for multipliers configured by DSP48s (built-in structures on the FPGA).

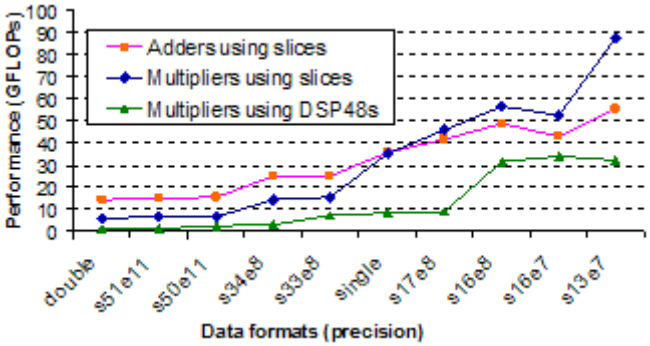

Figure 1: Lower-precision floating point achieves higher performance

2 BACKGROUND & RELATED WORK

Linear algebra is widely used in scientific computations. Therefore, tremendous effort has focused on improving linear algebra performance by building optimized libraries for various computer architectures [7][8]. Inspired by the great performance of lower-precision floating point components and previous research using low-precision or fixed-precision data formats, we seek to achieve high performance linear algebra on FPGAs without losing accuracy by using mixed-precision algorithms and architectures. This research is closely related to previous work with reconfigurable computing, linear algebra, and iterative refinement algorithms.

2.1 Reconfigurable Computers

We are interested in high performance reconfigurable computing because the traditional Von Neumann computer architecture is currently facing significant challenges. First, computers architects must invest more power and area on the cache hierarchy to bridge the widening gap between CPU and main memory performance. Meanwhile, heat dissipation problems caused by high clock rates make it increasingly difficult to increase the CPU frequency. Due to these reasons, computer architects struggle to fully utilize the exploding chip capacity brought by modern Integrated Circuit (IC) technology. In contrast, FPGAs have enjoyed substantial growth in performance and capacity [6]. Reconfigurable computers containing FPGAs have shown tens to hundreds of times speedup for many computationally intensive algorithms, such as linear algebra [1][2][7][14], random number generators [15], and molecular dynamics [16]. Hence, mixed-precision algorithms and architectures for reconfigurable computers show promise to assist in LU decomposition and linear direct solvers.

2.2 Linear Algebra on Reconfigurable Computers

Due to the intensive floating point computational loads and high communication to computation ratio of linear algebra applications, especially sparse matrix operations, traditional computers usually cannot achieve good performance. Utilizing FPGAs for high performance linear algebra computation has been explored with promising speedups achieved over CPUs. Previous work reveal that the performance and resource cost of linear algebra on FPGAs depend on the size, frequency, and pipeline latency of floating point IP cores [7][14].

Matrix factorization is widely used to solve linear equations, and LU decomposition is the most commonly used method for matrix factorization. Significant previous work addresses hardware architectures for this important computational kernel [14][17][18][19][20]. For example, Govindu developed a circular linear array architecture on FPGAs and achieved a 10% - 60% reduction in energy over that of a traditional CPU. Zhuo et al improved this design by increasing parallelism through more PEs and achieved higher GFLOPS performance than a 2.2 GHz AMD Opteron processor by using a Virtex-II Pro FPGA [14]. All these previous designs assume that target matrices are positive definite and no pivoting is required. Although pivoting complicates control logic, it increases the numeric stability of LU decomposition. Therefore, our hardware architecture supports pivoting.

Lower-precision designs can achieve higher performance by accommodating more floating point operators, increasing clock frequency, and shortening the pipeline latency. Memory bandwidth is another key factor affecting the performance of FPGA accelerators for linear algebra. For instance, the actual performance of sparse matrix computation [7] is usually limited by the memory bandwidth. Using lower-precision arithmetic directly enables higher performance for these designs due to reduced memory bandwidth needs. However, the solutions of linear solvers usually must meet certain accuracy constraints. We use lower-precision data formats internally (on the FPGA) for high performance and provide high precision externally (on the CPU). Our linear solver provides solutions as accurate as if only high-precision data formats are used but at a much faster speed.

2.3 Mixed-Precision Iterative Refinement

Iterative refinement algorithms were introduced in the 1960s [1]. While traditional algorithms use a specific data format, we use mixed-precision data formats to combine the high performance of lower-precision formats with the accuracy of higher-precision data formats. Buttari et al investigated single/double mixed-precision algorithms for linear solvers [5]. Their results show that mixed-precision algorithms can exploit single precision floating point performance and also achieve the double precision accuracy on several architectures, such as traditional CPUs and cell processors [5]. Strzodka et al discussed using mixed-precision algorithms for iterative solvers [4]. Their approach separated the loops of iterative solvers into two parts: lower precision inner loops and higher-precision outer loops. The results from inner loops are used as preconditions for the outer loops. They used software to emulate the possible inner and outer loops required by mixed-precision algorithms and compared the results to double precision algorithms. This previous research demonstrates the possibility of using mixed-precision algorithms but no previous configurable architectures have been built.

3 UNIQUENESS OF THE APPROACH

3.1 Mixed-Precision Direct Solver

The primary contribution of this work is exploring the performance of mixed-precision direct solvers on FPGAs. This is in contrast to [4], which used different data formats for iterative solver loops, required numerous refinement loops, and resulted in high execution

time for many test cases. Suppose matrix A can be factored as $PA=LU$ with partial pivoting, where L is a lower triangular matrix, U is an upper triangular matrix, and P is a permutation matrix used for pivoting. The direct solver with iterative refinement [5] is shown in Figure 2, where the first two steps are very similar to traditional direct solver algorithms and the refinement loops are taken to improve the accuracy based on the available solution. The iterative refinement process is similar to Newton's method applied to $f(x) = b - Ax$. The required number of refinement iterations depends on the matrix condition numbers and data precision. Table 1 gives the average number of iterations required for random matrices whose entries are generated by Gaussian random number generators [1]. In this analysis, we keep the same number of exponent bits as in double precision and vary the size of the mantissa. The number of iterations increases from right to left and from top to bottom, which corresponds to reduced precision and matrices with bigger condition numbers [1].

```
Factorize A to LU: PA=LU.
Solve LUx=Pb
while (r is too big & maximum loops not reached)
        r=b-Ax
        Solve Ly=Pr
        Solve Uz=y
        x=x+z
end
```

Figure 2: Iterative refinement direct solver algorithm

Table 1: Refinement iterations for different precision

Mantissa Bits / Problem Size	16	23	31	48	52
128	4	2	1	1	0
256	5.1	2.1	1	1	0
512	6.1	2.5	1	1	0
1024	6.3	2.6	1	1	0
2048	9.3	3	1.3	1	0
4096	13.3	3.1	1.43	1	0

The key idea here is that the factorization $PA=LU$ and the triangular solver $LUx=Pb$ are computed in lower precision; while the residual and solution update are computed in higher precision. This algorithm produces a solution correct to the higher-precision, provided matrix A is not too ill–conditioned [5]. The behavior of the mixed-precision method depends strongly on the accuracy with which the residual is computed [1]. The potential performance gain of using this algorithm comes from faster factorization computation, which is $O(n^3)$ and dominates the runtime of the algorithm. The other steps, including the triangular solver, residual computation, and solution update, are $O(n^2)$. Furthermore, shorter data formats usually reduce the memory bandwidth requirement.

3.2 Implementation on Cray XD1 Supercomputer

We realized a unique LU decomposition and mixed-precision direct solver on a reconfigurable supercomputer. The Cray XD1 supercomputer incorporates reconfigurable computing accelerators to deliver significant speedup of targeted applications. The basic architectural unit of the Cray XD1 system is the chassis, which can contain one to six compute blades. Each compute blade includes two 64-bit AMD Opteron processors configured as a two-way symmetric multiprocessor (SMP) that runs Linux. Each compute processor can be assigned 1 to 8 GB DDR. FPGAs can be included as coprocessors by adding an expansion module on the compute blade. As shown in Figure 3, the processors, FPGAs, and memory are linked within a chassis and between chasses by a high-speed fabric called the RapidArray interconnect. Besides the main memory, each FPGA module contains four QDR II SRAMs for high-speed storage. The Cray XD1 machine at ORNL (Tiger) has 12 chasses containing 144 Opteron processors and 6 Xilinx XC2VP50-7 FPGAs.

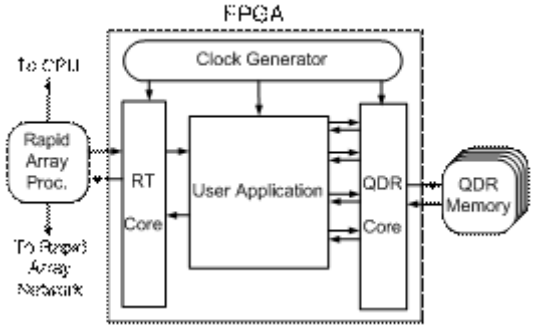

Figure 3: FPGA organization on Cray XD1 supercomputer [10]

The mixed-precision algorithm needs floating point components with different data formats contained in a single system. This can be realized by implementing different precision floating point IP cores on a single FPGA chip. An alternative approach is to have different precision arithmetic on different FPGAs, or on both FPGAs and CPUs. The former approach requires the FPGAs have enough capacity to accommodate parallel computational engines for multiple data formats. An issue with the second approach is that the data movement within FPGAs, or between FPGAs and CPUs, should not be too frequent to affect the performance. Considering the limited capacity of current FPGAs, our work uses the latter approach. Figure 4 shows our innovative mixed-precision hybrid

direct solver on the Cray-XD1 supercomputer. The lower-precision LU decomposer is mapped to the FPGA for high performance while the forward/backward solver and iterative refinements are mapped to the Opteron processor in higher-precision. The latter could also be implemented on FPGAs, but complicated control logic is required to achieve fine grained parallelism. Furthermore, the division operation of each iteration cannot be easily parallelized and requires an expensive floating point divider. On the other hand, the computational complexity for the operations on the CPU is only $O(n^2)$, while that for the LU decomposition on the FPGA is $O(n^3)$. For an n by n matrix, the computational load for LU decomposition is n times of that for forward and backward solvers.

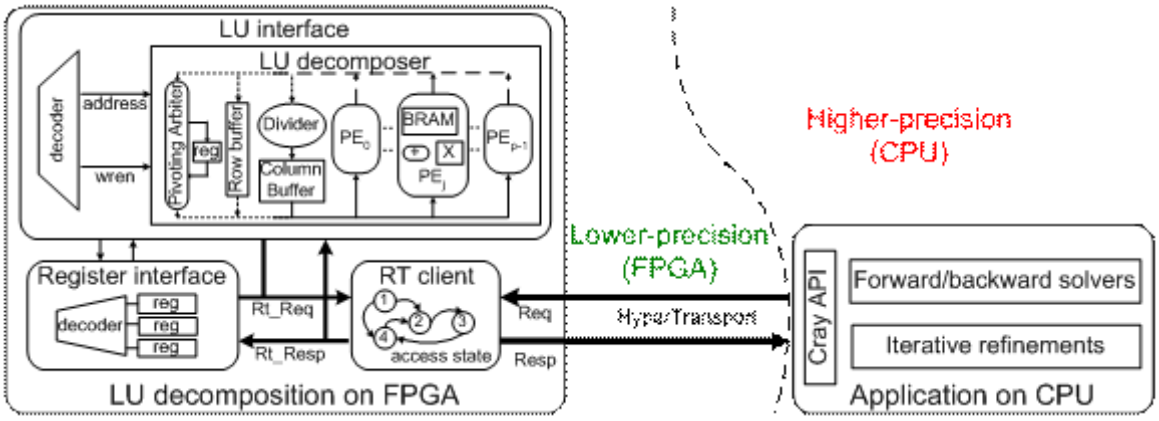

Figure 4: Hybrid mixed-precision direct solver on a reconfigurable supercomputer (Cray XD1)

To the best of our knowledge, our work is the first to design pivoting circuits for LU decomposition on FPGAs [1]**.** As shown in the left half of Figure 4, we build an LU decomposer with multiple parallel processing elements (PEs). The complete LU design is deeply pipelined with minimal overhead [1]. We improve the performance of the direct solver by accelerating LU decomposition on FPGAs. The maximum number of PEs and their local memory size are limited by the available resources of the targeted FPGA. The frequency shown in Table 2 is the frequency for our design running on the ORNL Cray XD1. Because more PEs can be implemented for lower-precision designs than for higher-precision designs on a FPGA, lower-precision designs can achieve higher performance.

Table 2: LU Implementation on Xilinx XC2VP50-7 FPGA

Design	Double (s52e11)	s31e8	s16e7
PE number	8	16	32
Max size	128	128	256
Achievable Frequency	120MHz	130MHz	140MHz
Slices	21044 (89%)	20356 (86%)	20907 (88%)
BRAMs	84 (36%)	84 (36%)	130 (56%)
MULT18X18	128 (55%)	64 (27%)	32(13%)

4 RESULTS AND CONTRIBUTIONS

4.1 Results

We compare the performance of our design with CPUs. Pivoting is supported for both FPGA and CPU implementations. The CPU performance is tested on a 2.2GHz Opteron processor in a Cray-XD1 supercomputer running ATLAS, an optimized BLAS library. For large matrices, which cannot be accommodated by FPGA on-chip memory, we propose to use QDR memory as in Figure 3. A clock cycle accurate model is built to predict the performance of our mixed-precision direct solver. As shown in Figure 5, lower-precision designs have higher performance by taking advantage of additional parallelism and higher frequency. The performance of the lower-precision design s16e7 is about 2 to 3 times better than the double precision design and CPUs.

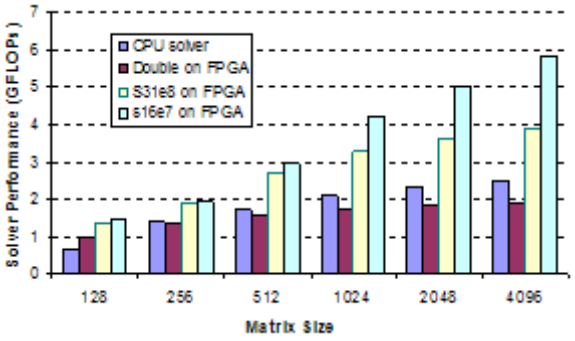

Figure 5: Performance of direct solvers

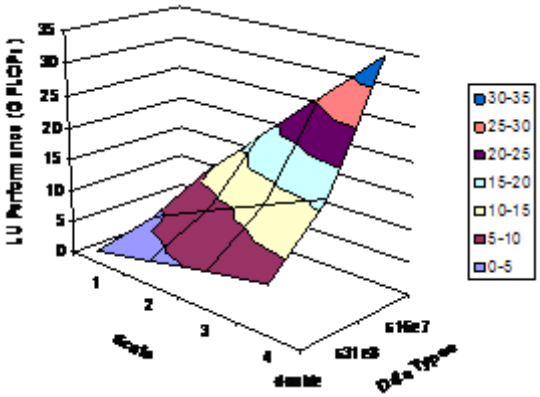

Figure 6: Prediction of LU performance

For the platform used with this research, the Cray XD1, the FPGA included on the nodes (Xilinx XC2VP50) uses older technology than the Opteron processors included on the nodes. Given the fact that the floating point performance of FPGAs increases much faster than CPUs [6], we expect the GFLOPs rate of newer FPGAs to result in even better speedups. For newer FPGAs, we are able to implement more computational logic at higher frequency. Figure 6 predicts the performance of our LU decomposer using our clock cycle accurate performance model [9], where "scale" describes a multiplicative scaling factor for the number dividers and PEs can be accommodated compared to the XC2VP50 FPGA on the Cray XD1. For example, with s16e7 data representation about 4 times as many divider units and PEs can be accommodated by a Xilinx XC4VLX200 FPGA contemporary to the Opteron processor. The vertical axis in Figure 6 shows a 35x performance gain of FPGAs over CPUs. Improved FPGA clock rates will further increase the LU decomposer performance.

4.2 Contributions

The work proposes innovative mixed-precision algorithms and architectures on reconfigurable computers for high performance linear algebra. To achieve this goal, we developed high performance circuits and algorithms on FPGAs. This research contributes both hardware/software implementations and theoretical derivations including:

- The first to explore the performance of mixed-precision linear direct solvers in configurable data formats.
- The first to develop high performance mixed-precision direct solvers on FPGAs.
- The first to develop a high performance hybrid linear direct solver that utilizes both a CPU and a FPGA.
- Implementation of a FPGA LU decomposition architecture with significant speedup over CPUs.
- The first to propose hardware architectures for pivoting algorithm on FPGAs using HDL code.
- The first to implement LU decomposition on a reconfigurable supercomputer and give performance analysis.

5 CONCLUSIONS

This paper presents a mixed-precision linear solver on FPGAs to achieve both the high performance of lower-precision arithmetic and the accuracy of higher-precision at the same time. To the best of our knowledge, this is the first mixed-precision architecture for linear solvers on hardware. Test results on the Cray XD1 supercomputer show that significant performance gains can be achieved by using our mixed-precision algorithm and architecture and therefore point out a new approach - mixed precision architecture - for future high performance computers and hardware accelerators. This research also brings many opportunities for future work, such as mixed-precision linear algebra library on hardware, mixed-precision high performance CPUs, and other mixed-precision algorithms.

6 ACKNOWLEDGEMENTS

This project was partially supported by the University of Tennessee Science Alliance and ORNL. Olaf Storaasli of ORNL was very helpful and provided excellent insight into solvers. I also would like to thank Jack Dongarra and Stanimire Tomov of the University of Tennessee for useful discussions on mixed-precision algorithms.

7 REFERENCES

[1] J. Sun, G. D. Peterson, O. O. Storaasli, "High Performance Mixed-Precision Linear Solver for FPGAs," *IEEE Transaction on Computers,* to appear.

[2] J. Sun, "Obtaining High Performance via Lower-Precision FPGA Floating Point Units," *Supercomputing*, Reno NV, Nov. 2007.

[3] H. Bowdler, R. Martin, G. Peters, and J. Wilkinson. "Linear Algebra: Solution of Real and Complex Systems of Linear Equations," *Numerische Mathematic*, 1966.

[4] R. Strzodka and D. Göddeke, "Mixed Precision Methods for Convergent Iterative Schemes," *EDGE*, North Carolina, May 2006.

[5] A. Buttari, J. Dongarra, J. Langou, J. Langou, P. Luszczek, and J. Kurzak, "Mixed Precision Iterative Refinement Techniques for the Solution of Dense Linear Systems," *International Journal of High Performance Computer Applications*, Volume 21, page 457-466, 2007.

[6] K. D. Underwood. "FPGAs vs. CPUs: Trends in Peak Floating-Point Performance," *FPGA*, Feb 2004.

[7] J. Sun, G. Peterson, O.O. Storaasli, "Sparse Matrix-vector Multiplication Design on FPGAs", *15th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM),* April, 2007.

[8] R. Whaley, A Petitet, and J Dongarra, "Automated Empirical Optimization of Software and the ATLAS Project," *Parallel Computing*, Volume 27, page 3-35, 2001.

[9] J. Sun, "High Performance Reconfigurable Computing for Linear Algebra: Design and Performance Analysis," PhD dissertation, the University of Tennessee, Knoxville, 2008.

[10] Cray Inc. http://www.cray.com

[11] Xilinx Inc. http://www.xilinx.com

[12] Gokul Govindu, Ronald Scrofano and Viktor K. Prasanna, "A Library of Parameterizable Floating-Point Cores for FPGAs and Their Application to Scientific Computing," *International Conference on Engineering of Reconfigurable Systems and Algorithms*, June 2005.

[13] X. Wang, M. Leeser, and H. Yu, "A Parameterized Floating-Point Library Applied to Multispectral Image Clustering," *Proceedings of the 7th annual MAPLD International Conference*, September 2004.

[14] Ling Zhuo, Viktor K. Prasanna, "High-Performance and Parameterized Matrix Factorization on FPGAs," *FPL*, Madrid, Spain, August 2006.

[15] Y. Bi, G. D. Peterson, L. Warren, and R. Harrison, "Hardware Acceleration of Parallel Lagged-Fibonacci Pseudo Random Number Generation," *ERSA*, June 2006.

[16] Y. Gu, T. VanCourt, and M. Herbordt. "Improved Interpolation and System Integration for FPGA-Based Molecular Dynamics Simulations," *FPL*, 2006.

[17] V. Daga, G. Govindu, S. Gangadharpalli, V. Sridhar, and V. K. Prasanna, "Efficient Floating-point Based Block LU Decomposition on FPGAs," *ERSA*, June 2004.

[18] Gokul Govindu, Seonil Choi, Viktor K. Prasanna, Vikash Daga, Sridhar Gangadharpalli, and V. Sridhar, "A High-Performance and Energy-efficient Architecture for Floating-point based LU Decomposition on FPGAs," *RAW*, April 2004.

[19] X. Wang and S.G. Ziavras, "Parallel LU Factorization of Sparse Matrices on FPGA-Based Configurable Computing Engines," *Concurrency Computation: Practice and Experience*, Vol. 16, No. 4, April 2004.

[20] DaeGon Kim, Sanjay V. Rajopadhye, "An Improved Systolic Architecture for LU Decomposition," *IEEE 17th International Conference on Application-specific Systems*, Architectures and Processors (ASAP), 2006.