2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines

Fixed Point Lanczos: Sustaining TFLOP-equivalent Performance in FPGAs for Scientific Computing

Juan L. Jerez
Department of Electrical and Electronic Engineering
Imperial College London
United Kingdom
jlj05@imperial.ac.uk

George A. Constantinides
Department of Electrical and Electronic Engineering
Imperial College London
United Kingdom
gac1@imperial.ac.uk

Eric C. Kerrigan
Department of Electrical and Electronic Engineering and Department of Aeronautics
Imperial College London
United Kingdom
e.kerrigan@imperial.ac.uk

***Abstract*—We consider the problem of enabling fixed-point implementations of linear algebra kernels to match the strengths of the field-programmable gate array (FPGA). Algorithms for solving linear equations, finding eigenvalues or finding singular values are typically nonlinear and recursive making the problem of establishing analytical bounds on variable dynamic range non-trivial. Current approaches fail to provide tight bounds for this type of algorithms. We use as a case study one of the most important kernels in scientific computing, the Lanczos iteration, which lies at the heart of well known methods such as conjugate gradient and minimum residual, and we show how we can modify the algorithm to allow us to apply standard linear algebra analysis to prove tight analytical bounds on all variables of the process, regardless of the properties of the original matrix. It is shown that the numerical behaviour of fixed-point implementations of the modified problem can be chosen to be at least as good as a double precision floating point implementation. Using this approach it is possible to get sustained FPGA performance very close to the *peak* general-purpose graphics processing unit (GPGPU) performance in FPGAs of comparable size when solving a single problem. If there are several independent problems to solve simultaneously it is possible to exceed the peak floating-point performance of a GPGPU, obtaining approximately 1, 2 or 4 TFLOPs for error tolerances of 10^{-7}, 10^{-5} and 10^{-3}, respectively, in a large Virtex 7 FPGA.**

***Keywords*-Field programmable gate arrays; Scientific computing; High performance computing; Iterative algorithms; Fixed-point arithmetic;**

I. Introduction

Acceleration of ubiquitous linear algebra operations such as solving systems of linear equations or determining the eigenvalues of a matrix has the potential to enable new applications in all areas of engineering and science. It is typically taken for granted that such algorithms require floating-point implementations in order to behave in a numerically reliable way due to the wide dynamic range of values that the variables can take.

Moore's law has continued to promote FPGAs to a level where it has become possible to provide substantial acceleration over microprocessors by directly implementing floating-point linear algebra kernels in FPGAs [1]–[5]. However, floating-point operations remain expensive to implement in FPGAs, mainly because there is no hard support in the FPGA fabric to facilitate the normalisation and denormalisation operations required before and after every floating-point addition or subtraction. This observation has led to the development of tools aimed towards fusing entire floating-point datapaths, reducing this overhead [6], [7]. Besides, there is also the ever-increasing competition of GPGPUs, which are easier to program and have much higher peak floating-point performance, even if sustained performance is often not comparable on real-world problems.

FPGAs excel at fixed-point computation due to hard integer support built into the FPGA fabric. For addition and multiplication, a reduction of more than one order of magnitude in both latency and amount of resources needed when migrating to similar sized fixed-point datapath mean that there is a potential performance improvement of two orders of magnitude by porting a multiply-accumulate intensive iterative floating-point application to fixed-point. However, most algorithms in scientific computing are nonlinear and recursive, which adds additional challenges. Existing methods for providing analytical bounds on signals in order to avoid overflow errors cannot handle this kind of computation effectively [8].

We propose a novel methodology for tackling this type of linear algebra kernel. By carefully modifying the algorithm using a novel form of preconditioning [9] and applying some mathematical techniques we can put tight bounds on all variables of the Lanczos iteration, which is one of the most widely used kernels in scientific computing, with applications ranging from line equation solution to eigenvalue computation.

The paper is organized as follows: Section II describes the Lanczos algorithm and a parameterizable custom computing architecture is proposed in Section III. The potential benefits of a fixed-point implementation over a floating-point implementation are also discussed. The methodology that we have used for bounding the worst-case peak values and reducing the dynamic range of variables is introduced in

978-0-7695-4699-5/12 $26.00 © 2012 IEEE
DOI 10.1109/FCCM.2012.19

Section IV, followed by numerical results showing that the numerical behaviour is practically identical as with double precision floating point, hence the FLOP-equivalence. In Section V we introduce the design automation tool that selects the number of bits used to represent each signal and the degree of parallelization such that the resulting circuit meets the numerical quality requirements and the limited resource constraints while minimizing latency. Section VI presents the relative performance comparison between the fixed-point and floating-point implementations in an FPGA and the absolute performance comparison against the peak performance of a high-end GPGPU.

II. The Lanczos Iteration

The Lanczos algorithm [10], described in Algorithm 1, transforms a symmetric matrix $A \in \mathbf{R}^{N \times N}$ into a tridiagonal matrix T (only the diagonal and off-diagonals are non-zero) with similar spectral properties as A using orthogonal transformation matrix Q. At every iteration the approximation is refined such that

$$Q_i^T A Q_i = T_i := \begin{bmatrix} \alpha_1 & \beta_1 & & 0 \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{i-1} \\ 0 & & \beta_{i-1} & \alpha_i \end{bmatrix}, \quad (1)$$

where $Q_i \in \mathbf{R}^{N \times i}$ and $T_i \in \mathbf{R}^{i \times i}$. The tridiagonal matrix T_i is easier to operate on than the original matrix. It can be used to extract the eigenvalues and singular values of A [11], or to solve systems of linear equations using the conjugate gradient (CG) [12] method when A is positive definite (all eigenvalues are positive) or the minimum residual (MINRES) [13] method when A is indefinite (can have positive and negative eigenvalues). Summarizing, this kernel is the key building block in modern iterative algorithms for solving the most important linear algebra problems involving symmetric matrices, hence there is considerable value in accelerating this computation.

Algorithm 1 Lanczos algorithm

Given q_1 such that $\|q_1\|_2 = 1$ and an initial value $\beta_0 := 1$
for $i = 1$ to i_{max} **do**
1. $q_i \leftarrow \frac{q_i}{\beta_{i-1}}$
2. $z_i \leftarrow Aq_i$
3. $\alpha_i \leftarrow q_i^T z_i$
4. $q_{i+1} \leftarrow z_i - \alpha q_i - \beta_{i-1} q_{i-1}$
5. $\beta_i \leftarrow \|q_{i+1}\|_2$

end for

Methods involving the Lanczos iteration are typically used for large sparse problems arising in scientific computing where direct methods, such as LU and Cholesky factorization or the Schur tridiagonal decomposition, cannot be used due to prohibitive memory requirements [14]. However, iterative methods have additional properties that also make them preferable for small problems arising in real-time applications, since they allow one to trade-off accuracy for computation time [3].

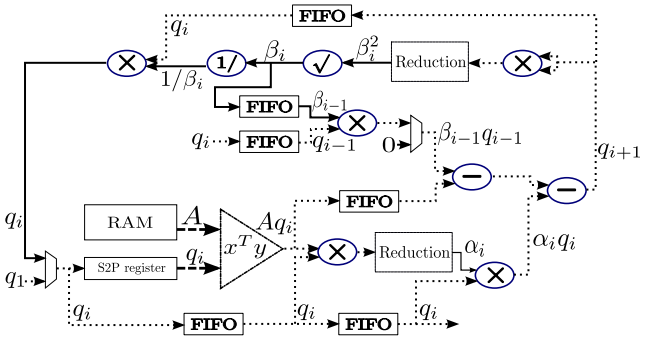


Figure 1: Lanczos compute architecture. Dotted lines denote links carrying vectors whereas solid lines denote links carrying scalars. The two thick dotted lines going into the $x^T y$ block denote N parallel vector links. The input to the circuit is q_1 going into the multiplexer and the matrix A being written into on-chip RAM. The output is α_i and β_i.

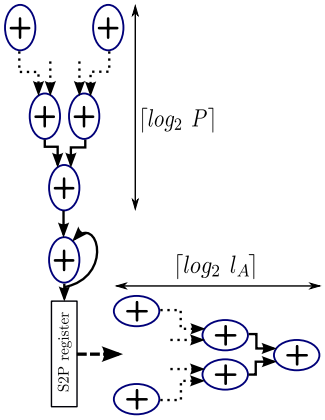


Figure 2: Reduction circuit. Uses $P + l_A - 1$ adders and a serial-to-parallel shift register of length l_A.

III. Parameterizable Architecture Generating Tool

We now describe our tool, which takes as inputs the data type, number of bits, level of parallelism and the latencies of an adder/subtracter (l_A), multiplier (l_M), square root (l_{SQ}) and divider (l_D) and automatically generates an architecture in VHDL.

The proposed compute architecture for implementing Algorithm 1 is shown in Figure 1. The most computationally intensive operation is the $\mathcal{O}(N^2)$ matrix-vector multiplication in line 2 of Algorithm 1. This is implemented in the block labelled $x^T y$, which is a parallel pipelined dot-product unit consisting of a parallel array of N multipliers followed

by an adder reduction tree of depth $\lceil \log_2 N \rceil$. The remaining operations are all $\mathcal{O}(N)$ vector operations and $\mathcal{O}(1)$ scalar operations that use dedicated components.

The amount of parallelism in the circuit is parameterized by parameter P. For instance, if $P = 2$, there will be two $x^T y$ blocks operating in parallel, one operating on the odd rows of A, the other on the even. All links carrying vector signals will branch into two links carrying the even and odd components, respectively, and all arithmetic units operating on vector links will be replicated. Note that the square root and divider, which consume most resources, only operate on scalar values, hence there will only be one of each of these units regardless of the value of P. For the memory subsystem, instead of having N independent memories each storing one column of A, there will be $2N$ independent memories, where half of the memories will store the even rows and the other half will store the odd rows of A.

The latency for one Lanczos iteration in terms of clock cycles is given by

$$L := \left\lceil \frac{N}{P} \right\rceil + l_A \lceil \log_2 N \rceil + 5 l_M + l_A + l_{SQ} + l_D + 2 + 2 l_{red}, \quad (2)$$

where

$$l_{red} := \left\lceil \frac{N}{P} \right\rceil + l_A \lceil \log_2 P \rceil + l_A + l_A \lceil \log_2 l_A \rceil - 1 \quad (3)$$

is the number of cycles it takes the reduction circuit, described in Figure 2, to reduce the incoming P streams to a single scalar value. This operation is necessary when computing $q_i^T A q_i$ and $q_{i+1}^T q_{i+1}$. Note in particular that for a fixed-point implementation where $l_A = 1$ and $P = 1$, the reduction circuit is a single adder and $l_{red} = N$, as expected.

A direct mapping of floating-point Lanczos-based algorithms into hardware has previously led to very significant acceleration over sequential microprocessor implementations [3], [4]. However, in FPGAs fixed-point multiplication and addition units use approximately one order of magnitude fewer resources and fewer clock cycles than floating-point units of the same number of bits. In addition, clock frequencies tend to be higher as a result of simpler circuitry. Figure 3 shows the latency of 32-bit fixed-point and single precision floating-point implementations of the Lanczos kernel. For a fixed-point implementation, smaller arithmetic latencies mean that the constant term in the latency expression has less weight, hence the incremental benefit of adding more parallelism is greater. Furthermore, a fixed-point implementation allows one to move further down the parallelism axis due to fewer resources being needed for individual arithmetic units. Larger problems benefit more from extra parallelism in all cases.

IV. Fixed-point Implementation

There are several challenges that need to be addressed before implementing an application in fixed-point. Firstly, one should determine the worst-case peak values for every signal in the circuit in order to avoid overflow errors. Secondly, the dynamic range of the signals should be constrained enough to allow numbers to be represented accurately with a reasonable number of bits. If these conditions are not satisfied, a fixed-point implementation will not provide a reliable result.

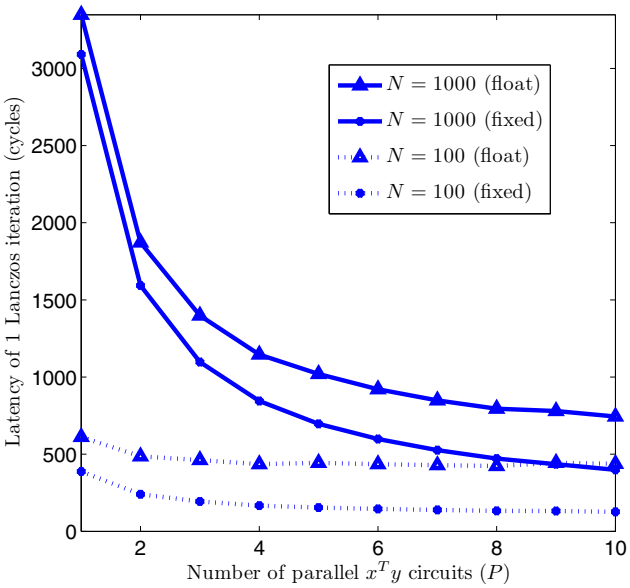


Figure 3: Latency of one Lanczos iteration for several levels of parallelism. For single precision floating-point: $l_A = 12$, $l_M = 8$, $l_{SQ} = l_D = 28$. For 32-bit fixed-point: $l_A = 1$, $l_M = 2$, $l_{SQ} = l_D = 28$.

For linear time-invariant (LTI) algorithms it is possible to use discrete-time system theory to put tight analytical bounds on worst-case peak values [15]. A linear algebra operation that meets such requirements is matrix-vector multiplication, where the input is a vector within a given range and the matrix does not change over time. For some nonlinear non-recursive algorithms interval arithmetic can be used to propagate data ranges forward through the computation graph [16]. Often this approach can be overly pessimistic for non-trivial graphs.

However, linear algebra kernels for solving systems of equations, finding eigenvalues or performing singular value decomposition are nonlinear and recursive. The Lanczos iteration belongs to this class. For this type of computation the bounds given by interval arithmetic quickly blow up, rendering useless information. As a consequence, these algorithms are typically implemented using floating-point arithmetic.

We propose to use a preconditioner [9] to modify the problem to make it suitable for fixed-point computation. For example, if we want to solve the system of linear equations

$Ax = b$, we propose instead to solve the problem

$$M^{-\frac{1}{2}} A M^{-\frac{1}{2}} y = M^{-\frac{1}{2}} b$$
$$\widehat{A} y = \widehat{b},$$

where M is chosen to ensure the absence of overflow in a fixed point implementation. The solution to the original problem can be recovered easily through the transformation $x := M^{-\frac{1}{2}} y$. Of course, the preconditioner $M^{\frac{1}{2}}$ should be easier to invert than the original matrix.

We employ the simplest invertible preconditioner, a positive diagonal matrix M whose diagonal elements are given by the following expression:

$$M_{kk} := \sum_{j=1}^{N} |A_{kj}|, \tag{4}$$

i.e. M_{kk} is the absolute sum of the elements in row k.

When using this preconditioner we have the following theorem proven in [17]:

Theorem 1. *Given preconditioner (4), the symmetric Lanczos algorithm applied to $\widehat{A} := M^{-\frac{1}{2}} A M^{-\frac{1}{2}}$, for any symmetric matrix A, has intermediate variables with the following bounds for all i:*

- $q_i \in (-1, 1)$
- $\widehat{A} \in (-1, 1)$
- $\widehat{A} q_i \in (-1, 1)$
- $\alpha_i \in (-1, 1)$
- $\beta_i q_{i-1} \in (-1, 1)$
- $\alpha_i q_i \in (-1, 1)$
- $\widehat{A} q_i - \beta_{i-1} q_{i-1} \in (-2, 2)$
- $q_{i+1} \in (-1, 1)$
- $q_{i+1}^T q_{i+1} \in (-1, 1)$
- $\beta_i \in (\epsilon, 1)$
- $\frac{1}{\beta_i} \in (1, 1/\epsilon)$

where ϵ is a small positive number determined by the termination criterion.

Theorem 1 implies that by employing preconditioner (4) we can put tight analytical bounds on all variables and we can constrain signals to a very small dynamic range enabling potentially reliable fixed-point implementations. The following corollary is a consequence of the theorem.

Corollary 1. *For the integer part of a fixed-point 2's complement representation we require, including the sign bit, one bit for q_i, $\widehat{A}$, $\widehat{A} q_i$, α_i, $\beta_i q_{i-1}$, $\alpha_i q_i$, q_{i+1}, β_i and $q_{i+1}^T q_{i+1}$, two bits for $\widehat{A} q_i - \beta_{i-1} q_{i-1}$, and $\lceil -\log_2(\epsilon) \rceil$ bits for $\frac{1}{\beta_i}$.*

Preconditioners are traditionally used to accelerate the convergence of numerical algorithms rather than to bound their dynamic range [9]. In this work we are using a preconditioner in a novel way: to allow us to put tight bounds on variables, however, we have observed that preconditioner (4) also improves the speed of convergence for a large number of problems. This is a nice side-benefit, although we cannot claim that the preconditioner will improve convergence in all cases, because a universal preconditioner does not exist [14].

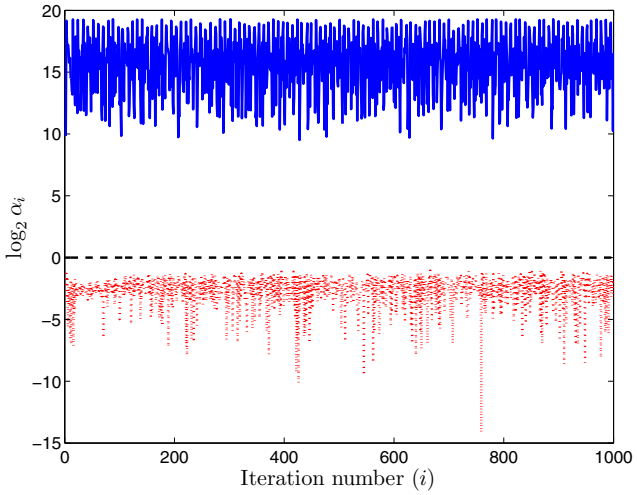


Figure 4: Peak values of α during the Lanczos process with (dotted) and without (solid) a preconditioner for benchmark problem `c-18` from [18].

The use of the preconditioner is essential for a reliable fixed-point implementation – without it variable peak values and dynamic range are much larger and the absence of overflow cannot be guaranteed. To illustrate this point Figure 4 shows the evolution of the peak value of α for preconditioned and non-preconditioned Lanczos on one of the example problems used in the following section.

A. Numerical Results

We now present the results of our numerical tests which will be used in Section V to determine the necessary number of bits to achieve the target level of accuracy in the solution.

In order to evaluate the numerical behaviour of the Lanczos method using the proposed approach we examine the convergence of the MINRES algorithm described in Algorithm 2. This algorithm attempts to minimize the 2-norm of the residual of the modified problem $\|\widehat{A} y - \widehat{b}\|_2$, which is related to the residual of the original problem through $\|M^{-\frac{1}{2}}(Ax - b)\|_2$. Since our primary objective is to evaluate the numerical behaviour of the computationally-intensive Lanczos kernel, the operations outside Lanczos are carried out in double precision floating point.

Figure 5 shows the convergence behaviour of a double precision floating point implementation and several fixed-point implementations for a symmetric matrix from the University of Florida sparse matrix collection [18]. The fixed-point implementations have practically the same numerical behaviour as the double precision implementation. At some point, the accuracy cannot be improved further. The point

Algorithm 2 MINRES algorithm

Given q_i, α_i and β_i from Algorithm 1 and initial values $\gamma_1 := 1$, $\gamma_0 := 1$, $\sigma_1 := 0$, $\sigma_0 := 0$, $\eta = 1$, $w_0 := 0$, $w_{-1} := 0$ and $y := 0$:

for $i = 1$ to ... i_{max} **do**

$\delta \leftarrow \gamma_i \alpha_i - \gamma_{i-1} \sigma_i \beta_{i-1}$

$\rho_1 \leftarrow \sqrt{\delta + \beta_i^2}$

$\rho_2 \leftarrow \sigma_i \alpha_i + \gamma_{i-1} \gamma_i \beta_{i-1}$

$\rho_3 \leftarrow \sigma_{i-1} \beta_{i-1}$

$\gamma_{i+1} \leftarrow \frac{\delta}{\rho_1}$

$\sigma_{i+1} \leftarrow \frac{\beta_i}{\rho_1}$

$w_i \leftarrow \frac{q_i - \rho_3 w_{i-2} - \rho_2 w_{i-1}}{\rho_1}$

$y \leftarrow y + \gamma_{i+1} \eta w_i$

$\eta \leftarrow -\sigma_{i+1} \eta$

end for

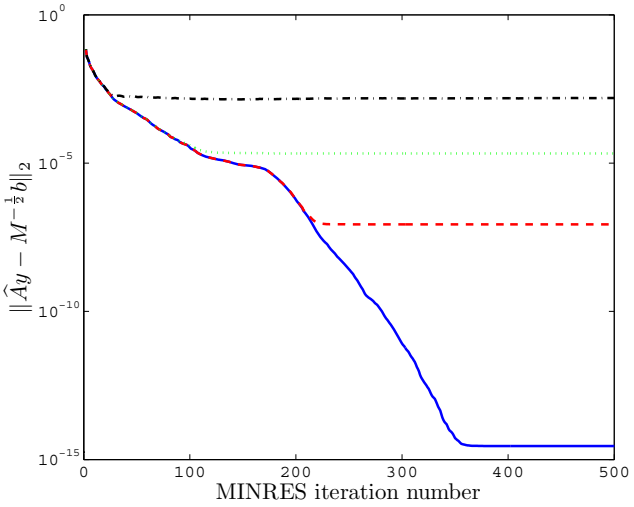


Figure 5: Convergence results when solving a linear system using MINRES for `sherman1` from [18] ($N = 1000$). The solid line represents the double precision floating-point implementation, whereas the dashed, dotted and dash-dotted lines represent fixed-point implementations with $k = 32$, 24 and 16 bits for the fractional part of signals, respectively.

at which this occurs is dependent on the machine precision. The behaviour is similar for all linear systems for which the MINRES algorithm converges to the solution in double precision floating-point arithmetic.

The final attainable accuracy is determined by the machine unit round-off. In fact, if e_k and u_k are the attainable values of $\|\widehat{A}y - \widehat{b}\|_2$ and the unit round-off, respectively, for a specific implementation with k bits used to represent the fractional part of numbers we have

$$e_k \propto u_k \tag{5}$$

for all k, and when using floor rounding $u_k := 2^{-k}$. The constant of proportionality C, is different for each

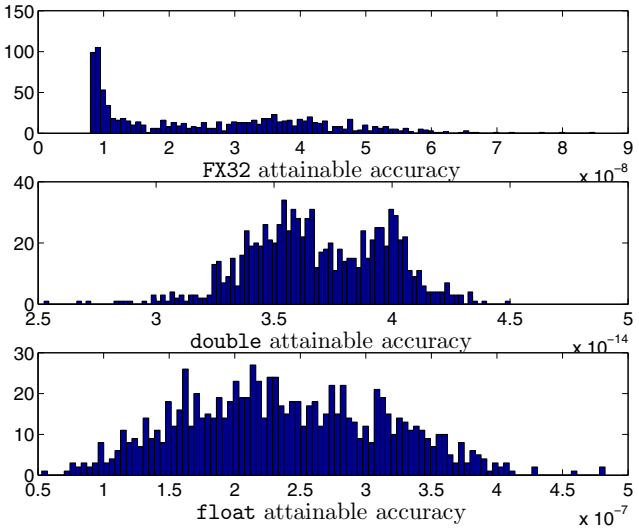


Figure 6: Histogram of the final attainable error for several implementations.

problem and it captures the numerical difficulty of the problem in a single number. It cannot be computed *a priori* but expression (5) allows one to determine the attainable accuracy for any number of bits from the result of a single simulation run. An upper bound estimate of C that can be computed *a priori* is the condition number of the matrix.

In order to investigate the variation in C within a single application we constructed a benchmark set of approximately 1000 linear systems coming from an optimal controller for a Boeing 747 model [19] under many different operating conditions. The linear systems were solved using a $k = 32$ fixed-point implementation, and single and double precision floating-point implementations, and the attainable accuracy was recorded in each case. The results are shown in Figure 6. Even though the benchmark set contains matrices with condition numbers ranging from 50 to 10^{10} we can see that the variation in the attainable accuracy, and therefore C, is much smaller. This is because the great majority of the eigenvalues follow a similar distribution for all matrices, with only a few eigenvalues becoming very large in some cases. This is likely to be true also in other application areas.

V. Resource Constrained Framework

In order to evaluate the performance of our designs for a given target FPGA chip we created a design automation tool that generates optimum designs with respect to the following rule:

$$\min_{P,k} \; L(P,k)$$

subject to

$$\mathbb{P}(e_k \leq \eta) > 90\% \tag{6}$$

$$R(P,k) \leq \text{FPGA}_{\text{area}}, \tag{7}$$

Table I: Resource usage

Type	Amount
Adder/Subtracter	$P(N+3)+2l_A-2$
Multiplier	$P(N+5)$
Divider	1
Square root	1
Memory - $2\lceil\frac{N}{P}\rceil$ elements	PN
Memory - N elements	$5P$

where $L(P,k)$ is defined in (2) with the explicit dependence of latency on the number of fraction bits, k, noted. $\mathbb{P}(e_k \leq \eta)$ represents the probability that any problem chosen at random from the benchmark set meets the user-specified accuracy constraint η, and is used to model the fact that for any finite precision representation – fixed point or double precision floating point – there will be problem instances that fail to converge for numerical reasons. The user can specify η – the tolerance on the error, and the number of problems allowed to fail to converge – here 10%. $R(P,k)$ is a vector representing FFs, LUTs, embedded multipliers and embedded RAM utilization for an architecture with parallelism degree P (see Section III) and a k-bit fixed point datapath.

This problem can be easily solved. First, determine the minimum number of fraction bits k necessary to satisfy the accuracy requirements (6) by making use of the information in Figure 6 and (5). Once k is fixed, find the maximum P such that (7) remains satisfied using the information in Table I and a model for the number of LUTs, flip-flops and embedded multipliers necessary for implementing each arithmetic unit for different number of bits and data representations [20]. If $P=1$ is not able to satisfy (7), then the problem is infeasible and either the accuracy requirements have to be relaxed or a larger FPGA will be necessary.

For implementations with a small number of bits memory is typically the limiting factor, whereas for larger numbers of bits embedded multipliers limit the amount of parallelism. In the former case, storage of some of the columns of A is implemented with registers so flip-flops become the limiting resource. In the latter case, some multipliers are implemented using LUTs so these become the limiting resource. Figure 7 shows the trade-off between latency and FFs offered by the floating-point implementations and two fixed-point implementations that meet the same accuracy requirements as single and double precision floating-point. The trade-off is similar for other resources. We can see that the fixed-point implementations make better utilization of the available resources to reduce latency by providing the same solution quality.

VI. Performance Results

In this section we will evaluate the relative performance of the fixed-point and floating-point implementations under the resource constraint framework of Section V for a Virtex 7 XT 1140 [21]. Then we will evaluate the absolute performance of the fixed-point implementations against a GPGPU with a peak performance of 1 TFLOP/s.

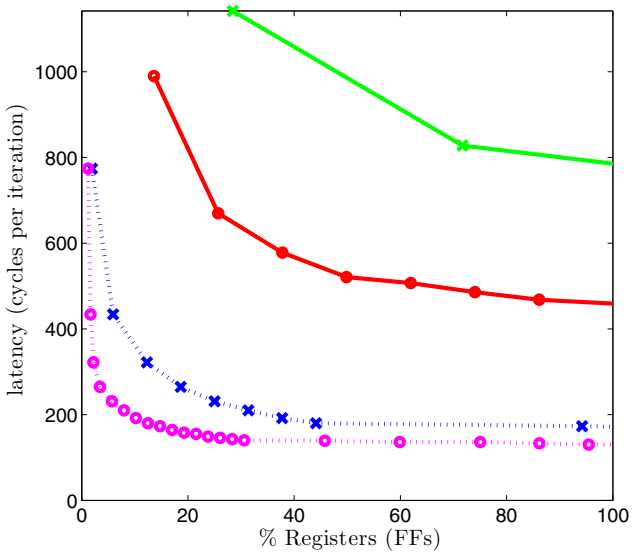


Figure 7: Latency tradeoff against FF utilization (from model) on a Virtex 7 XT 1140 [21] for $N=229$. Double precision ($\eta = 4.05\times10^{-14}$) and single precision ($\eta = 3.41\times10^{-7}$) are represented by solid lines with crosses and circles, respectively. Fixed-point implementations with $k=53$ and 29 are represented by the dotted lines with crosses and circles, respectively. These implementations match the accuracy requirements of the floating-point implementations.

The trade-off between latency (2) and accuracy requirements for our FPGA implementations is investigated in Figure 8. For high accuracy requirements a large number of bits are needed reducing the extractable parallelism and increasing the latency. As the accuracy requirements are relaxed it becomes possible to reduce latency by increasing parallelism. The figure also shows that the fixed-point implementations provide a better trade-off even when the accuracy of the calculation is considered.

The simple control structures in our design and the pipelined arithmetic units allow the circuits to be clocked at frequencies up to 400MHz. Noting that each Lanczos iteration requires $2N^2+8N$ operations we plot the number of operations per second (OP/s) against accuracy requirements in Figure 9. For extremely high accuracy requirements, not attainable by double precision floating-point, the fixed-point implementation still achieves approximately 100 GOP/s. For accuracy requirements of 10^{-6} and 10^{-3} the fixed-point implementations can achieve approximately 200 and 300 GOP/s, respectively. For some real-time applications, it might be tolerable to use solutions with an accuracy of 10^{-3} if the result can be applied fast enough. However, $N=229$ is a relatively small number and the incremental benefit of

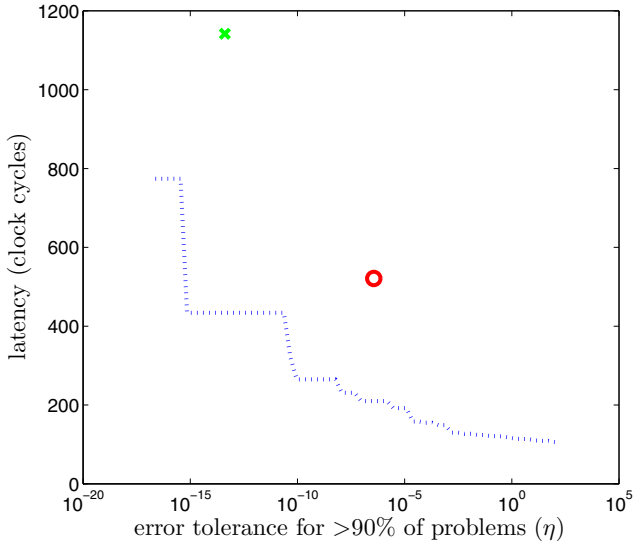


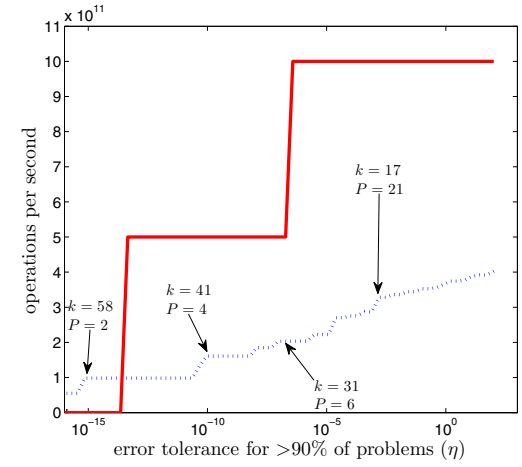
Figure 8: Latency tradeoff against accuracy requirements on a Virtex 7 XT 1140 [21] for $N = 229$. The dotted line, the cross and the circle represent fixed-point and double and single precision floating-point implementations, respectively.

extra parallelization is rather small, limiting the resulting performance. Figure 9 (b) shows that if the matrix is slightly larger it is possible to improve performance significantly and approach the peak GPGPU performance curve. For the GPGPU it is assumed that the double precision performance is half of the single precision performance. It should be noted that for this type of computation, the actual GPGPU performance will be significantly lower than its peak.

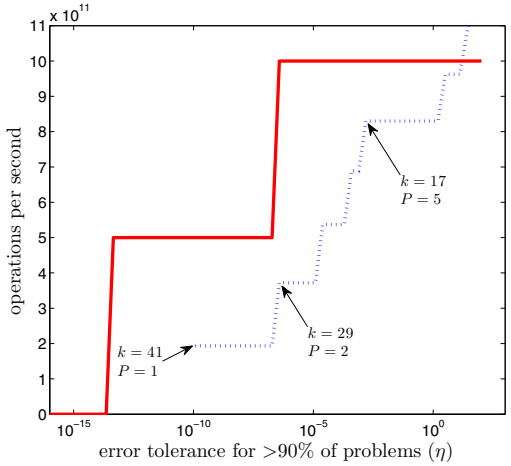
The results presented so far have assumed that we are processing a single problem at a time. Using this approach the arithmetic units in our circuit are always idle for some fraction of the iteration time. In the situation when there are many independent problems available [22], it is possible to multiplex these problems into the same circuit to hide the pipeline latency and keep arithmetic units busy. If the extra storage needed does not hinder parallelism, then it is possible to achieve much higher performance, exceeding the peak GPGPU performance for most accuracy requirements even for small problems, as shown in Figure 9 (c).

VII. Conclusion

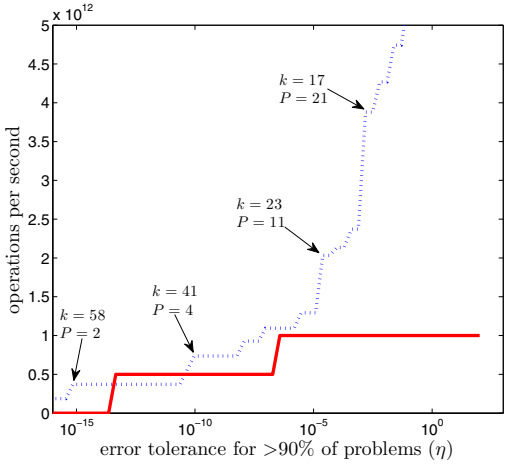
Porting an application from a floating-point implementation to a fixed-point implementation in an FPGA can provide substantial performance improvements due to the hard support for this type of computation in the FPGA fabric. However, fixed-point computation presents additional challenges from the design point of view. We have shown that by modifying the problem and employing linear algebra analysis it is possible to tackle issues associated with reliable fixed-point implementations in situations where techniques



(a) $N = 229$, single problem



(b) $N = 750$, single problem



(c) $N = 229$, many problems

Figure 9: Sustained computing performance for fixed-point implementations on a Virtex 7 XT 1140 [21] for different accuracy requirements. The solid line represents the peak performance of a 1 TFLOP/s GPGPU.

such as interval arithmetic fail. This methodology has allowed us to implement the Lanczos iteration, an ubiquitous operation in scientific computing, in fixed-point while guaranteeing that overflow errors wont occur. The flexibility provided by the configurable nature of FPGAs allows us to choose the right number of bits for each signal to meet the accuracy requirements and achieve sustained performance rivalling peak GPGPU performance.

Future work could attempt to extend this methodology to other linear algebra kernels. For instance, it should be relatively straightforward to extend this ideas to the Arnoldi iteration, which can operate on non-symmetric matrices.

ACKNOWLEDGMENT

This work was supported by the EPSRC (grants EP/G031576/1 and EP/I012036/1) and the EC FP7 project EMBOCON. The authors would also like to acknowledge fruitful discussions with Mr. Theo Drane and Mr Ammar Hasan.

REFERENCES

[1] K. D. Underwood and K. S. Hemmert, "Closing the gap: CPU and FPGA trends in sustainable floating-point BLAS performance," in *In Proc. 12th IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, CA, USA, Apr 2004, pp. 219–228.

[2] W. Zhang, V. Betz, and J. Rose, "Portable and scalable FPGA-based acceleration of a direct linear system solver," in *In Proc. Int. Conf. on Field Programmable Technology*, Taipei, Taiwan, Dec 2008, pp. 17–24.

[3] D. Boland and G. A. Constantinides, "An FPGA-based implementation of the MINRES algorithm," in *Proc. Int. Conf. on Field Programmable Logic and Applications*, Heidelberg, Germany, Sep 2008, pp. 379–384.

[4] A. R. Lopes and G. A. Constantinides, "A high throughput FPGA-based floating-point conjugate gradient implementation," in *Proc. 4th Int. Workshop on Applied Reconfigurable Computing*, London, UK, Mar 2008, pp. 75–86.

[5] Y. Dou, Y. Lei, G. Wu, S. Guo, J. Zhou, and L. Shen, "FPGA accelerating double/quad-double high precision floating-point applications for exascale computing," in *In Proc. 24th ACM Int. Conf. on Supercomputing*, Tsukuba, Japan, Jun 2010, pp. 325–335.

[6] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, Aug 2011.

[7] M. Langhammer and T. VanCourt, "FPGA floating point datapath compiler," in *In Proc. 17th IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, CA, USA, Apr 2007, pp. 259–262.

[8] G. A. Constantinides, N. Nicolici, and A. B. Kinsman, "Numerical data representations for FPGA-based scientific computing," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 8–17, Aug 2011.

[9] A. Greenbaum, *Iterative Methods for Solving Linear Systems*, 1st ed., ser. Frontiers in Applied Mathematics. Philadelphia, PA, USA: Society for Industrial Mathematics, 1987, no. 17.

[10] C. Lanczos, "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators," *Journal of Research of the National Bureau of Standards*, vol. 45, no. 4, pp. 255–282, Oct 1950.

[11] G. Golub and W. Kahan, "Calculating the singular values and pseudo-inverse of a matrix," *SIAM Journal on Numerical Analysis*, vol. 2, no. 2, pp. 205–224, 1965.

[12] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, pp. 409–436, Dec 1952.

[13] C. C. Paige and M. A. Saunders, "Solution of sparse indefinite systems of linear equations," *SIAM Journal on Numerical Analysis*, vol. 12, no. 4, pp. 617–629, Sep 1975.

[14] G. H. Golub and C. F. V. Loan, *Matrix Computations*, 3rd ed. Baltimore, USA: The Johns Hopkins University Press, 1996.

[15] S. K. Mitra, *Digital Signal Processing*, 3rd ed. New York, USA: McGraw-Hill, 2005.

[16] A. Benedetti and P. Perona, "Bit-width optimization for configurable DSP's by multi-interval analysis," in *In Proc. 34th Asilomar Conf. on Signals, Systems and Computers*, Pasadena, CA, USA, Nov 2000, pp. 355–359.

[17] J. L. Jerez, G. A. Constantinides, and E. C. Kerrigan, "Fixed-point Lanczos with guaranteed variable bounds," *SIAM Journal on Matrix Analysis and Applications*, (to be submitted).

[18] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Transactions on Mathematical Software*, (to appear). [Online]. Available: http://www.cise.ufl.edu/research/sparse/matrices

[19] C. V. D. Linden, H. Smaili, A. Marcos, G. Balas, D. Breeds, S. Runhan, C. Edwards, H. Alwi, T. Lombaerts, J. Groeneweg, R. Verhoeven, and J. Breeman. (2011) GARTEUR RECOVER benchmark. [Online]. Available: http://www.faulttolerantcontrol.nl/

[20] (2011) LogiCORE IP floating-point operator v5.0. Xilinx. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/floating_point_ds335.pdf

[21] (2011) Virtex-7 family overview. Xilinx. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

[22] J. L. Jerez, K.-V. Ling, G. A. Constantinides, and E. C. Kerrigan, "Model predictive control for deeply pipelined field-programmable gate array implementation: algorithms and circuitry," *IET Control Theory and Applications*, p. (accepted for publication), 2012.