A Low Complexity Scaling Method for the Lanczos Kernel in Fixed-Point Arithmetic

Juan L. Jerez, *Student Member, IEEE,* George A. Constantinides, *Senior Member, IEEE,* and Eric C. Kerrigan, *Member, IEEE*

Abstract—We consider the problem of enabling fixed-point implementation of linear algebra kernels on low cost embedded systems, as well as motivating more efficient computational architectures for scientific applications. Fixed-point arithmetic presents additional design challenges compared to floating-point arithmetic, such as having to bound peak values of variables and control their dynamic ranges. Algorithms for solving linear equations or finding eigenvalues are typically nonlinear and iterative, making solving these design challenges a non-trivial task. For these types of algorithms the bounding problem cannot be automated by current tools. We focus on the Lanczos iteration, the heart of well-known methods such as conjugate gradient and minimum residual. We show how one can modify the algorithm with a low complexity scaling procedure to allow us to apply standard linear algebra to derive 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 floating-point implementation, if necessary. The approach is evaluated on field-programmable gate array (FPGA) platforms, highlighting orders of magnitude potential performance and efficiency improvements by moving form floating-point to fixed-point computation.

Index Terms—Computer Arithmetic, Computations on Matrices, Numerical Algorithms, Design Aids

1 Introduction

The Lanczos iteration [1] is the key building block in modern iterative numerical methods for computing eigenvalues or solving systems of linear equations involving symmetric matrices. These methods are typically used in scientific computing applications, for example when solving large sparse linear systems of equations arising from the discretization of partial differential equations (PDEs) [2]. In this context, iterative methods are preferred over direct methods, because they can be easily parallelized and they can better exploit the sparsity in the problem to reduce computation and, perhaps more importantly, memory requirements [3]. However, these methods are also interesting for small- and medium-scale problems arising in real-time embedded applications. In the embedded domain, on top of the advantages previously mentioned, iterative methods allow one to trade off computation time for accuracy in the solution and enable the possibility of terminating the method early to meet real-time deadlines. An example application is real-time optimal decision making with applications in, for instance, advanced control systems [4].

In both cases more efficient forms of computation, in the form of new computational architectures and algorithms that allow for more efficient architectures, could enable new applications in many areas of science and engineering. In high-performance computing (HPC), power consumption is quickly becoming the key limiting factor for building the next generation of computing machines [5]. In embedded computing, cost, power consumption, computation time and size constraints often limit the complexity of the algorithms that can be implemented, thereby limiting the capabilities of the embedded solution.

Porting floating-point algorithm implementations to fixed-point arithmetic is an effective way to address these limitations. Because fixed-point numbers do not require mantissa alignment, the circuitry is significantly simpler and faster. The smaller delay in arithmetic operations leads to lower latency computation and shorter pipelines. The smaller resource requirements either result in more performance through parallelism for a given silicon budget, or a reduction in silicon area and lower power consumption and cost. This latter observation is especially important, since the cost of chip manufacturing increases at least quadratically with silicon area [6]. It is for this reason that fixed-point architectures are ubiquitous in high-volume low-cost embedded platforms, hence any new solution based on increasingly complex sophisticated algorithms must be able to run on fixed-point architectures to achieve high-volume adoption. In the HPC domain, heterogeneous architectures that incorporate fixed-point processing could help to lessen the effects of the power wall, which is the major hurdle in the

- *J. L. Jerez and G. A. Constantinides are with the Department of Electrical and Electronic Engineering at Imperial College London, SW7 2AZ, United Kingdom.*
  E-mail: jlj05@imperial.ac.uk
- *E. C. Kerrigan is with the Department of Aeronautics and the Department of Electrical and Electronic Engineering at Imperial College London, SW7 2AZ, United Kingdom.*

road to exascale computing [7].

However, while fixed-point arithmetic is widespread for simple digital signal processing (DSP) operations, it is typically assumed that floating-point arithmetic is necessary for solving general linear systems or general eigenvalue problems, due to the potentially large dynamic range in the data and consequently on the algorithm variables. Furthermore, the Lanczos iteration is known to be sensitive to numerical errors [8], hence moving to fixed-point arithmetic could potentially worsen the problem and lead to unreliable numerical behaviour.

To be able to take advantage of the simplicity of fixed-point circuitry and reduce cost, power and computation time, the complexity burden shifts to the algorithm design process [9]. In order to have a reliable fixed-point implementation one has to be able to establish bounds on all variables of the algorithm to avoid online shifting, which would negate any speed advantages, and avoid overflow errors. In addition, the bounds should be of the same order to minimize loss of precision when using constant wordlengths. There are several tools in the design automation community for handling this task [10]. However, because the Lanczos iteration is a nonlinear iterative algorithm, all state-of-the-art bounding tools fail to provide practical bounds. Unfortunately, most linear algebra kernels (except extremely simple operations) are of this type and they suffer from the same problem.

1.1 Summary of Contribution

We propose a novel scaling procedure first introduced by us in [11] to tackle the fixed-point bounding problem for the nonlinear and recursive Lanczos kernel. The procedure gives tight bounds for all variables of the Lanczos process regardless of the properties of the original matrix, while minimizing the computational overhead. The proof, based on linear algebra, makes use of the fact that the scaled matrix has all eigenvalues inside the unit circle. This kind of analysis is currently well beyond the capabilities of state-of-the-art automatic methods [12]. We then discuss the validity of the bounds under finite precision arithmetic and give simple guidelines to be used with existing error analysis [8] to ensure that the absence of overflow is maintained under inexact computation. We also show how the same scaling approach can be used for bounding variables in other nonlinear recursive linear algebra kernels based on matrix-vector multiplication, such as the Arnoldi iteration [13] – a generalization of the Lanczos kernel for non-symmetric matrices.

The potential efficiency improvements of the proposed approach are evaluated on an FPGA platform. While 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 [14]–[17], floating-point operations remain expensive to implement, 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 [18], [19]. However, the hard integer support built into the FPGA fabric means that the FPGA is a good example platform where the efficiency gap between fixed-point and floating-point computation is very large, which is why they are used for evaluation in this paper, even though the presented results are equally applicable for implementing these algorithms on embedded microcontrollers and fixed-point DSPs.

To exploit the architecture flexibility in an FPGA we present a parameterizable architecture generator where the user can tune the level of parallelization and the data type of each signal. This generator is embedded in a design automation tool that selects the best architecture parameters to minimize latency, while satisfying the accuracy specifications of the application and the FPGA resources available. Using this tool we show that it is possible to get sustained FPGA performance very close to the *peak* theoretical general-purpose graphics processing unit (GPGPU) performance when solving a single Lanczos problem to equivalent accuracy. If there are multiple independent problems to solve simultaneously it is possible to exceed the peak floating-point performance of a GPGPU. If one considers the power consumption of both devices, the fixed-point Lanczos solver on the FPGA is more than an order of magnitude more efficient than the peak GPGPU efficiency. The test data are obtained from a benchmark set of equations generated by from a Boeing 747 optimal controller.

1.2 Outline

The remainder of the paper is organised as follows: Section 2 describes the Lanczos algorithm; Section 3 presents our scaling procedure and contains the analysis to guarantee the absence of overflow in the Lanczos process; Section 4 presents numerical results showing that the numerical quality of the linear equation solution does not suffer by moving to fixed-point arithmetic; in Section 5 we introduce an FPGA design automation tool that generates minimum latency architectures given accuracy specifications and resource constraints; in Section 6 this tool is used to evaluate the potential relative performance improvement between fixed-point and floating-point FPGA implementations and perform an absolute performance comparison against the peak performance of a high-end GPGPU; Section 7 discusses the possibility of extending this methodology to other nonlinear re-

Algorithm 1 Lanczos algorithm

```
Require: Initial iterate $r_1$ such that $\|r_1\|_2 = 1$, $q_0 := 0$
    and $\beta_0 := 1$.
1: for $i = 1$ to $i_{\max}$ do
2:     $q_i \leftarrow \frac{r_i}{\beta_{i-1}}$
3:     $z_i \leftarrow Aq_i$
4:     $\alpha_i \leftarrow q_i^T z_i$
5:     $r_{i+1} \leftarrow z_i - \alpha_i q_i - \beta_{i-1} q_{i-1}$
6:     $\beta_i \leftarrow \|r_{i+1}\|_2$
7: end for
8: return $q_i$, $\alpha_i$ and $\beta_i$
```

cursive kernels based on matrix-vector multiplication. Section 8 concludes the paper.

2 The Lanczos Kernel

The Lanczos 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 an orthogonal transformation matrix Q. The method is described in Algorithm 1, where q_i is the i^{th} column of 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 [20], or to solve systems of linear equations of the form $Ax = b$ using the conjugate gradient (CG) [21] method when A is positive definite or the minimum residual (MINRES) [22] method when A is indefinite. The Arnoldi iteration, a generalisation of Lanczos for non-symmetric matrices, is used in the generalized minimum residual (GMRES) method for general matrices and is dicussed in Section 7. The Lanczos (and Arnoldi) algorithms account for the majority of the computation in these methods – they are the key building blocks in modern iterative algorithms for solving formulations of linear systems appearing in all kinds of applications.

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, cannot be used due to prohibitive memory requirements [23]. However, iterative methods have additional properties that also make them good candidates for small problems arising in real-time applications, since they allow one to trade-off accuracy for computation time [24].

3 Fixed-Point Analysis

A floating-point number consists of a sign bit, an exponent and a mantissa. The exponent value moves the binary point with respect to the mantissa. The dynamic range – the ratio of the smallest to largest representable number – grows doubly exponentially with the number of exponent bits, therefore it is possible to represent a wide range of numbers with a relatively small number of bits. However, because two operands can have different exponents it is necessary to perform denormalisation and normalisation operations before and after every addition or subtraction, leading to greater resource usage and longer delays. In contrast, fixed-point numbers have a fixed number of bits for the integer and fraction fields, i.e. the exponent does not change with time and it does not have to be stored. Fixed-point hardware is the same as for integer arithmetic, hence the circuitry is simple and fast, but the representable dynamic range is limited. However, if the dynamic range in the data is also limited and fixed, a 32-bit fixed-point processor can provide more precision than a 32-bit floating-point processor because there are no bits wasted for the exponent.

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 variable in order to avoid overflow errors. 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 [25]. 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 [26] can be used to propagate data ranges forward through the computation graph [27]. Often this approach can be overly pessimistic for non-trivial graphs because it cannot take into account the correlation between variables.

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. Table 1 highlights the limitations of state-of-the-art bounding tool Gappa [12] – a tool based on interval arithmetic – for handling the bounding problem for one iteration of Algorithm 1. Even when only one iteration is considered, the bounds quickly become impractical as the problem size grows, because the tool cannot use any extra information in matrix A beyond bounds on the individual coefficients. Other recent tools [28] that can take into account the correlation between input variables can help to tighten the single iteration bounds, but there is still a significant amount of conservatism. More complex tools [29] that can take into account additional prior information on the input variables can further improve the tightness of the bounds. However, as shown in Table 1, the complexity of the procedure limits its usefulness to very small problems. In addition, the bounds given by all these tools will grow further for more than one iteration. As a consequence, these types of algorithms are typically implemented using floating-point arithmetic because the absence of overflow errors cannot be guaranteed, in general, with a practical number of fixed-point bits for practical problems.

Despite the acknowledged difficulties there have been several fixed-point implementations of nonlinear recursive linear algebra algorithms. CG-like algorithms were implemented in [30], [31], whereas the Lanczos algorithm was implemented in [32]. Bounds on variables were established through simulation-based studies and adding a heuristic safety factor. In the targeted DSP applications, the types of problems that have to be processed do not change significantly over time, hence this approach might be satisfactory, especially if the application is not safety critical. In other applications, such as in optimization solvers, the range of linear algebra problems that need to be solved on the same hardware is so varied that it is not possible to assign wordlengths based on simulation in a practical manner. Importantly, in safety-critical applications analytical guarantees are desirable, since overflow errors can lead to catastrophic behaviour of the system [33].

3.1 A Scaling Procedure for Bounding Variables

We propose the use of a diagonal scaling matrix M to redefine the problem in a new co-ordinate system to allow us to control the bounds in all variables, such that the same fixed precision arithmetic can efficiently handle problems with a wide range of matrices. For example, if we want to solve the symmetric system of linear equations $Ax = b$, where $A = A^T$, we propose instead to solve the problem

$$MAMy = Mb$$
$$\Leftrightarrow \hat{A}y = \hat{b}\,,$$

where

$$\hat{A} := MAM,$$
$$\hat{b} := Mb\,,$$

and the elements of the diagonal matrix M are chosen as

$$M_{kk} := \frac{1}{\sqrt{\sum_{j=1}^{N} |A_{kj}|}} \tag{2}$$

to ensure the absence of overflow in a fixed-point implementation. The solution to the original problem can be recovered easily through the transformation $x = My$.

TABLE 1: Bounds on r_2 computed by state-of-the-art bounding tools [12], [28] given $r_1 \in [-1, 1]$ and $A_{ij} \in [-1, 1]$. The tool described in [29] can also use the fact that $\sum_{j=1}^{N} |A_{ij}| = 1$. Note that r_1 has unit norm, hence $\|r_1\|_\infty \leq 1$, and A can be trivially scaled such that all coefficients are in the given range. '-' indicates that the tool failed to prove any competitive bound. Our analysis will show that when all the eigenvalues of A have magnitude smaller than one, $\|r_i\|_\infty \leq 1$ holds independent of N for all iterations i.

N	2	4	10	100	150
$\|r_2\|_\infty$ – [12]	6	20	110	10100	22650
$\|r_2\|_\infty$ – [28]	4	16	100	10000	22500
$\|r_2\|_\infty$ – [29]	2	12	100	-	-
Runtime for [29] (seconds)	4719	0.5 *	29232	-	-

*No other bound smaller than $\|r_2\|_\infty \leq 12$ could be proved in less than two days.

An important point is that the scaling procedure and the recovery of the solution still have to be computed using floating-point arithmetic, due to the potentially large dynamic range and unboundness in the problem data. However, since the scaling matrix is diagonal, the cost of these operations is comparable to the cost of one iteration of the Lanczos algorithm. Since many iterations are typically required, most of the computation is still carried out in fixed-point arithmetic.

In order to illustrate the need for the scaling procedure, Figure 1 shows the evolution of the range of values of α_i (Line 4 in Algorithm 1) throughout the solution of one optimization problem from the benchmark set described in Section 4. Notice that a different Lanczos problem has to be solved at each iteration of the optimization solver. Since the range of Lanczos problems that have to be solved on the same hardware is so diverse, without using the scaling matrix (2) it is not possible to decide on a fixed data format that can represent numbers efficiently for all problems. Based on the simulation results, with no scaling one would need to allocate 22 bits for the integer part to be able to represent the largest value of α_i occurring in this benchmark set. Furthermore, using this number of bits would not guarantee that overflow will not occur on a different set of problems. The situation is similar for all other variables in the algorithm.

Instead, when using the scaling matrix (2) we have the following results:

Lemma 1. *The scaled matrix $\hat{A} := MAM$ with M as in (2) has, for any non-singular symmetric matrix A, spectral radius $\rho(\hat{A}) \leq 1$.*

Proof: Let $R_k := \sum_{j \neq k}^{N} |A_{kj}|$ be the absolute sum of the off-diagonal elements in row k, and let $D(A_{kk}, R_k)$ be a Gershgorin disc with centre A_{kk} and radius R_k. Consider an alternative non-symmetric preconditioned matrix $\widetilde{A} := M^2 A$. The absolute row

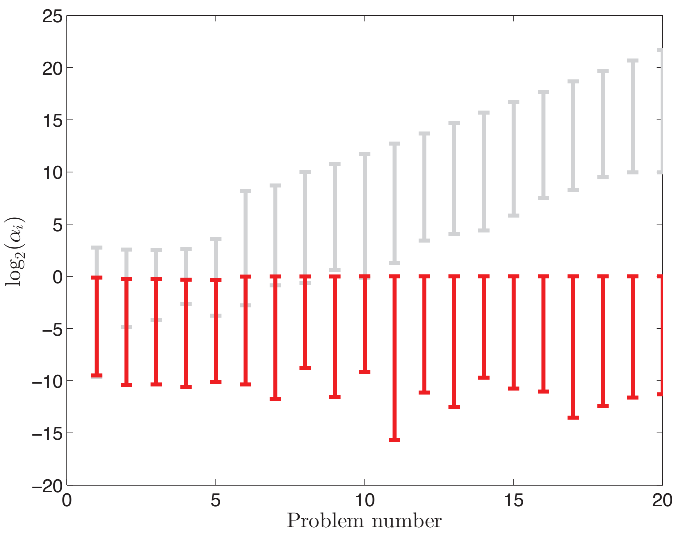


Fig. 1: Evolution of the range of values that α takes for different Lanczos problems arising during the solution of one optimization problem from the benchmark set of problems described in Section 4. The red solid and grey shaded curves represent the scaled and unscaled algorithms, respectively.

sum is equal to 1 for every row of $\widetilde{A}$, hence the Gershgorin discs associated with this matrix are given by $D(\widetilde{A}_{kk}, 1-|\widetilde{A}_{kk}|)$. It is straightforward to show that these discs always lie inside the interval between 1 and -1 when $|\widetilde{A}_{kk}| \leq 1$, which is the case here. Hence, $\rho(\widetilde{A}) \leq 1$ according to Geshgorin's circle theorem [23, Theorem 7.2.1]. Now, for an arbitrary eigenvalue-eigenvector pair (λ, v),

$$M^2Av = \lambda v \quad (3)$$
$$\Leftrightarrow MAv = M^{-1}\lambda v \quad (4)$$
$$\Leftrightarrow MAMu = \lambda u\,, \quad (5)$$

where (5) is obtained by substituting Mu for v. This shows that the eigenvalues of the non-symmetric preconditioned matrix $\widetilde{A}$ and the symmetric preconditioned matrix $\hat{A}$ are the same. The eigenvectors are different but this does not affect the bounds, which we derive next. □

Theorem 1. *Given the scaling matrix (2), the symmetric Lanczos algorithm applied to $\hat{A}$, for any non-singular symmetric matrix A, has intermediate variables with the following bounds for all i, j and k:*

- $[q_i]_k \in [-1, 1]$
- $[\hat{A}]_{kj} \in [-1, 1]$
- $[\hat{A}q_i]_k \in [-1, 1]$
- $\alpha_i \in [-1, 1]$
- $[\beta_{i-1}q_{i-1}]_k \in [-1, 1]$
- $[\alpha_i q_i]_k \in [-1, 1]$
- $[\hat{A}q_i - \beta_{i-1}q_{i-1}]_k \in [-2, 2]$
- $[r_{i+1}]_k \in [-1, 1]$
- $r_{i+1}^T r_{i+1} \in [0, 1]$
- $\beta_i \in [0, 1]$,

where i denotes the iteration number and $[]_k$ and $[]_{kj}$ denote the k^{th} component of a vector and kj^{th} component of a matrix, respectively.

Corollary 1. *For the integer part of a fixed-point 2's complement representation we require, including the sign bit, two bits for q_i, $\hat{A}$, $\hat{A}q_i$, α_i, $\beta_{i-1}q_{i-1}$, $\alpha_i q_i$, r_{i+1}, β_i and $r_{i+1}^T r_{i+1}$, and three bits for $\hat{A}q_i - \beta_{i-1}q_{i-1}$. Observe that the elements of M can be reduced by an arbitrarily small amount to turn the closed intervals of Theorem 1 into open intervals, saving one bit for all variables except for q_i.*

Proof of Theorem 1: The normalisation step in Line 2 of Algorithm 1 ensures that the Lanczos vectors q_i have unit norm for all iterations, hence all the elements of q_i are in $[-1, 1]$.

We follow by bounding the elements of the coefficient matrix:

$$|\hat{A}_{kj}| = M_{kk}M_{jj}|A_{kj}| \leq \frac{1}{\sqrt{|A_{kj}|}}\frac{1}{\sqrt{|A_{kj}|}}|A_{kj}| = 1, \quad (6)$$

where (6) follows from the definition of M.

Using Lemma 1 we can put bounds on the rest of the intermediate computations in the Lanczos iteration. We start with $\hat{A}q_i$, which is used in Lines 3, 4 and 5 in Algorithm 1:

$$\|\hat{A}q_i\|_\infty \leq \|\hat{A}q_i\|_2 \leq \|\hat{A}\|_2 = \rho(\hat{A}) \leq 1 \quad \forall i\,, \quad (7)$$

where (7) follows from the properties of matrix norms and the fact that $\|q_i\|_2 = 1$. The equality follows from the 2-norm of a real symmetric matrix being equal to its largest absolute eigenvalue [23, Theorem 2.3.1].

We continue by bounding α_i and β_i, which are used in Lines 2, 4, 5 and 6 of Algorithm 1 and represent the coefficients of the tridiagonal matrix described in (1). The eigenvalues of the tridiagonal approximation matrix (1) are contained within the eigenvalues of $\hat{A}$, even throughout the intermediate iterations [23, §9.1]. Hence, one can use the following relationship [23, §2.3.2]

$$\max_{jk} |[T_i]_{jk}| \leq \|T_i\|_2 = \rho(T_i) \leq \rho(\hat{A}) \leq 1 \quad \forall i \quad (8)$$

to bound the coefficients of T_i in (1), i.e. $|\alpha_i| \leq 1$ and $|\beta_i| \leq 1$, for all iterations.

Interval arithmetic can be used to show that the elements of $\alpha_i q_i$ and $\beta_{i-1}q_{i-1}$ are also between 1 and -1 and the elements of $\hat{A}q_i - \beta_{i-1}q_{i-1}$ are in $[-2, 2]$.

The following equality

$$Aq_i - \alpha_i q_i - \beta_{i-1}q_{i-1} = \beta_i q_{i+1} =: r_{i+1} \quad \forall i\,, \quad (9)$$

which always holds in the Lanczos process [23, §9], can be used to bound the elements of the auxiliary vector r_{i+1} in $[-1, 1]$ via interval arithmetic on the expression $\beta_i q_{i+1}$. We also know that β_i is non-negative so its bound derived from (8) can be refined.

Finally, we can also bound the intermediate computation in Line 6 of Algorithm 1 using

$$\|r_{i+1}\|_2 = |\beta_i|\|q_{i+1}\|_2 = |\beta_i| \leq 1 \quad \forall i\,,$$

hence $r_{i+1}^T r_{i+1}$ lies in $[0, 1]$. □

The following points should also be considered for a reliable fixed-point implementation of the Lanczos process:

- Division and square root operations are implemented as iterative procedures in digital architectures. The data types for the intermediate variables can be designed to prevent overflow errors. In this case, the fact that $[r_{i+1}]_k \leq \beta_i$ and $r_{i+1}^T r_{i+1} \leq 1$ can be used to establish tight bounds on the intermediate results for any implementation. For instance, all the intermediate variables in a CORDIC square root implementation [34] can be upper bounded by one if the input is known to be smaller than one.
- A possible source of problems both for fixed-point and floating-point implementations is encountering $\beta_i = 0$. However, this would mean that we have already computed a perfect tridiagonal approximation to $\hat{A}$, i.e. the roots of the characteristic polynomial of T_{i-1} are the same as those of the characteristic polynomial of T_i, signalling completion of the Lanczos process.
- If an upper bound estimate for $\rho(A)$ is available, it is possible to bound all variables analytically without using the scaling matrix (2). However, the bounds will lose uniformity, i.e. the elements of q_i would still be in $[-1, 1]$ but the elements of Aq_i would be in $[-\rho(A), \rho(A)]$.

The scaling operation that has been suggested in this section is also known as diagonal preconditioning. However, the primary objective of the scaling procedure is not to accelerate the convergence of the iterative algorithm, the objective of standard preconditioning. Sophisticated preconditioners attempt to increase the clustering of eigenvalues. Our scaling procedure, which has the effect on normalising the 1-norm of the rows of the matrix, can be applied after a traditional accelerating preconditioner. However, since this will move the eigenvalues, it cannot be guaranteed that the scaling procedure will not have a negative effect on the convergence rate. In cases where the convergence rate is not satisfactory, a better strategy could be to include the objective of normalising the 1-norm of the rows of the matrix in the design of the accelerating preconditioner to trade-off the goals of fast convergence and implementation cost. Since the design of sophisticated preconditioners is very application-specific it is not possible to describe a general procedure to achieve this goal.

3.2 Validity under Inexact Computations

We now use Paige's error analysis of the Lanczos process [8] to adapt the previously derived bounds in the presence of finite precision computations. We are interested in the worst-case error in any component. In the following, we will denote with ϵ_x the deviation of variable x from its value under exact arithmetic.

Unlike with floating-point arithmetic, fixed-point addition and subtraction operations involve no round-off error, provided there is no overflow and the result has the same number of fraction bits as the operands [35], which will be assumed in this section. For multiplication, the exact product of two numbers with k fraction bits can be represented using $2k$ fraction bits, hence a k-bit truncation of a 2's complement number incurs a round-off error bounded from below by -2^{-k}. Recall that in 2's complement arithmetic, truncation incurs a negative error both for positive and negative numbers.

The maximum absolute component-wise error in the variables involved in Algorithm 1 is summarised in the following proposition:

Proposition 1. *When using fixed-point arithmetic with k fraction bits and assuming no overflow errors, the maximum difference in the variables involved in the Lanczos process described by Algorithm 1 with respect to their exact arithmetic values can be bounded by:*

$$\|\epsilon_{q_i}\|_\infty \leq (N+4)2^{-k+1}\,, \tag{10}$$

$$\|\epsilon_{z_i}\|_\infty \leq \rho(\hat{A})\|\epsilon_{q_i}\|_\infty + N2^{-k}\,, \tag{11}$$

$$\|\epsilon_{\alpha_i}\|_\infty \leq \|\epsilon_q\|_\infty + \|\epsilon_{z_i}\|_\infty + N2^{-k}\,, \tag{12}$$

$$\|\epsilon_{r_i}\|_\infty \leq \rho(\hat{A})(N+7)2^{-k}\,, \tag{13}$$

$$\|\epsilon_{\beta_i}\|_\infty \leq 2\|\epsilon_{r_i}\|_\infty + N2^{-k}\,. \tag{14}$$

for all iterations i, where $\hat{A} \in \mathbf{R}^{N\times N}$.

Proof: In the original analysis presented in [8], higher order error terms are ignored since every term is assumed to be significantly smaller than one for the analysis to be valid, hence, higher order terms have a negligible impact on the final results. We do the same here as it significantly clarifies the presentation.

According to [8], the departure from unit norm in the Lanczos vectors can be bounded by

$$|q_i^T q_i - 1| \leq (N+4)2^{-k} \tag{15}$$

for all iterations i. In the worst case, all the error can be attributed to the same element in q_i, hence, neglecting higher-order terms we have

$$\|2\epsilon_{q_i}\|_\infty \leq (N+4)2^{-k}$$

leading to (10).

The error in Line 3 of Algorithm 1 can be written, using the properties of matrix and vector norms, as

$$\|\epsilon_{z_i}\|_\infty \leq \|\hat{A}\|_2\|\epsilon_{q_i}\|_2 + N2^{-k}\,,$$

where the last term represents the maximum component-wise error in matrix-vector multiplication. Using (15) one can infer that the bound on $\|\epsilon_{q_i}\|_2$ is the same as the bound on $\|\epsilon_{q_i}\|_\infty$ given by (10), leading to (11).

Neglecting higher order terms, the error in α_i in Line 4 of Algorithm 1 can be obtained by forward error analysis as

$$\|\epsilon_{\alpha_i}\|_\infty \leq \|z_i\|_\infty \|\epsilon_{q_i}\|_\infty + \|q_i\|_\infty \|\epsilon_{z_i}\|_\infty + N2^{-k} ,$$

where the last term arises from the maximum round-off error in the dot-product computation. Using the bounds given by Theorem 1 one arrives at (12).

Going back to the original analysis in [8] one can use the fact that the 2-norm of the error in the relationship (9), i.e.

$$\|Aq_i - \alpha_i q_i - \beta_{i-1} q_{i-1} - \beta_i q_{i+1}\|_2$$

can be bounded from below by

$$\rho(\hat{A})(N+7)2^{-k} \tag{16}$$

for all iterations i. One can infer that the bound on $\|\epsilon_{r_i}\|_2$ is the same as (16). Using the properties of vector norms leads to (13).

The error in Line 6 of Algorithm 1 is given by

$$\|\epsilon_{\beta_i}\|_\infty \leq 2\|r_i\|_\infty \|\epsilon_{r_i}\|_\infty + N2^{-k} .$$

Using the bounds given by Theorem 1 yields (14). □

The error bounds given by Proposition 1 enlarge the bounds given in Theorem 1. In order to prevent overflow in the presence of round-off errors the integer bitwidth for q_i has to increase by $\lceil \log_2((N+4)2^{-k-1}) \rceil$ bits, which will be one bit in all practical cases. For the remaining variables, which have bounds that depend on $\rho(\hat{A})$, one has two possibilities – either use extra bits to represent the integer part according to the new larger bounds, or adjust the spectral radius $\rho(\hat{A})$ through the scaling matrix (2) such that the original bounds still apply under finite precision effects.

The latter approach is likely to provide effectively tighter bounds. We now outline a procedure, described in the following lemma, for controlling $\rho(\hat{A})$ and give an example showing how to make use of it.

Lemma 2. *If each element of the scaling matrix (2) is multiplied by $(1+\varepsilon)$, where ε is a small positive number, the scaled matrix $\hat{A} := MAM$ has, for any non-singular symmetric matrix A, spectral radius $\rho(\hat{A}) \leq \frac{1}{1+\varepsilon}$.*

Proof: We now have $|\widetilde{A}_{kk}| \leq \frac{1}{1+\varepsilon}$. The new Gershgorin discs are given by $D(\widetilde{A}_{kk}, \frac{1}{1+\varepsilon} - |\widetilde{A}_{kk}|)$, which can be easily proved to lie inside the interval between $-\frac{1}{1+\varepsilon}$ and $\frac{1}{1+\varepsilon}$. □

For instance, if one decides to use $k = 20$ fraction bits on the benchmark problems described in Section 4 with dimension $N = 229$, the worst error bounds would be given by

$$\|\epsilon_\alpha\|_\infty \leq 4.4 \times 10^{-4}[\rho(\hat{A}) + 1] + 4.37 \times 10^{-4} ,$$
$$\|\epsilon_\beta\|_\infty \leq 4.5 \times 10^{-4}\rho(\hat{A}) + 2.18 \times 10^{-4} .$$

The value of ϵ in Lemma 2 is chosen such that the following inequalities are satisfied

$$\rho(\hat{A}) + \|\epsilon_\alpha\|_\infty \leq 1 ,$$
$$\rho(\hat{A}) + \|\epsilon_\beta\|_\infty \leq 1 ,$$

which, for the given values, is satisfied by $\varepsilon \geq 0.0013$.

Algorithm 2 MINRES algorithm

```
Require: Initial values $\gamma_1 := 1$, $\gamma_0 := 1$, $\sigma_1 := 0$, $\sigma_0 := 0$, $\zeta = 1$, $w_0 := 0$, $w_{-1} := 0$ and $y := 0$. Given $q_i$, $\alpha_i$ and $\beta_i$ from Algorithm 1:
1: for $i = 1$ to $i_{max}$ do
2:     $\delta_i \leftarrow \gamma_i \alpha_i - \gamma_{i-1}\sigma_i\beta_{i-1}$
3:     $\rho_{i,1} \leftarrow \sqrt{\delta_i^2 + \beta_i^2}$
4:     $\rho_{i,2} \leftarrow \sigma_i\alpha_i + \gamma_{i-1}\gamma_i\beta_{i-1}$
5:     $\rho_{i,3} \leftarrow \sigma_{i-1}\beta_{i-1}$
6:     $\gamma_{i+1} \leftarrow \frac{\delta_i}{\rho_{i,1}}$
7:     $\sigma_{i+1} \leftarrow \frac{\beta_i}{\rho_{i,1}}$
8:     $w_i \leftarrow \frac{q_i - \rho_{i,3}w_{i-2} - \rho_{i,2}w_{i-1}}{\rho_{i,1}}$
9:     $y \leftarrow y + \gamma_{i+1}\zeta w_i$
10:    $\zeta \leftarrow -\sigma_{i+1}\zeta$
11: end for
12: return $y$
```

4 Numerical Results

In this section we show that even though these algorithms are known to be vulnerable to round-off errors [36] they can still be executed using fixed-point arithmetic reliably by using our proposed approach.

In order to evaluate the numerical behaviour of the Lanczos method, we examine the convergence of the Lanczos-based MINRES algorithm, described in Algorithm 2, which solves systems of linear equations by minimising the 2-norm of the residual $\|\hat{A}y_i - \hat{b}\|_2$. Notice that in order to bound the solution vector y, one would need an upper bound on the spectral radius of $\hat{A}^{-1}$, which depends on the minimum absolute eigenvalue of $\hat{A}$. In general it is not possible to obtain a lower bound on this quantity, hence the solution update cannot be bounded and has to be computed using floating-point arithmetic. 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 arithmetic.

Figure 2 shows the convergence behaviour of a single precision floating point implementation and several fixed-point implementations for a symmetric matrix from the University of Florida sparse matrix collection [37]. All implementations exhibit the same convergence rate. There is a difference in the final attainable accuracy due to the accumulation of round-off errors, which is dependent on the precision used. The figure also shows that the fixed-point implementations have a stable numerical behaviour, i.e. the accumulated round-off error converges to a finite value. The numerical behaviour is similar for all linear systems for which the MINRES algorithm converges to the solution in double precision floating-point arithmetic.

In order to investigate the variation in attainable accuracy for a larger set of problems we created a benchmark set of approximately 1000 linear systems,

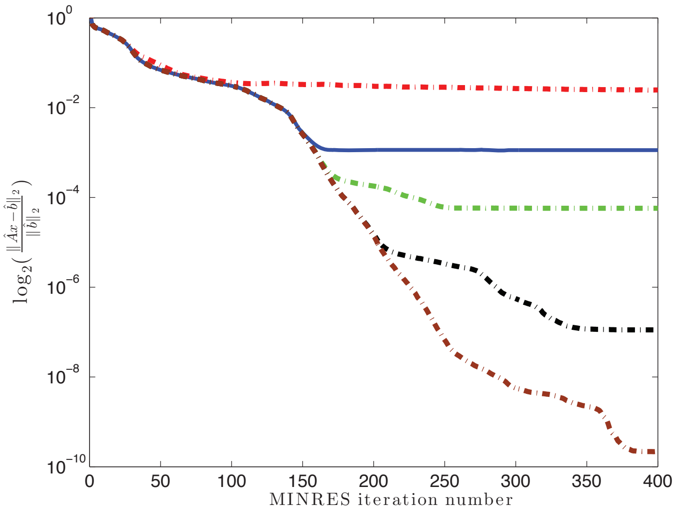


Fig. 2: Convergence results when solving a linear system using MINRES for benchmark problem `sherman1` from [37] with $N = 1000$ and condition number 2.2×10^4. The solid line represents the single precision floating-point implementation (32 bits including 23 mantissa bits), whereas the dotted lines represent, from top to bottom, fixed-point implementations with $k = 23, 32, 41$ and 50 bits for the fractional part of signals, respectively.

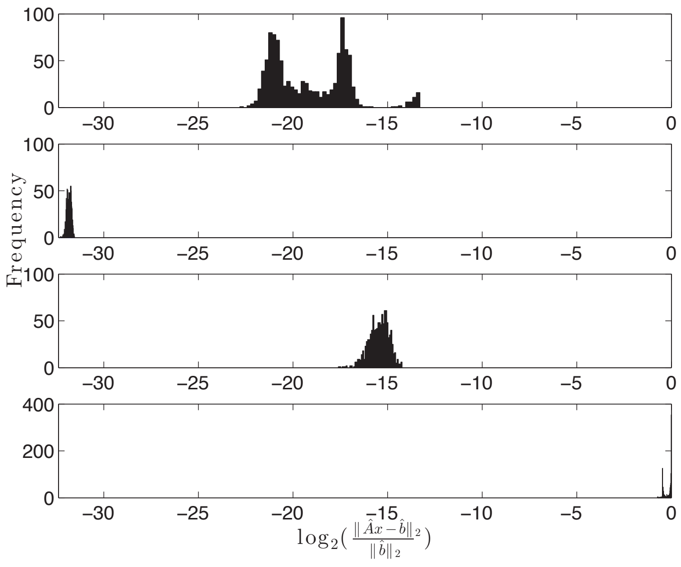


Fig. 3: Histogram showing the final log relative error $\log_2(\frac{\|\hat{A}x-\hat{b}\|_2}{\|\hat{b}\|_2})$ at termination for different linear solver implementations. From top to bottom, preconditioned 32-bit fixed-point, double precision floating-point and single precision floating-point implementations, and unpreconditioned single precision floating-point implementation.

coming from an optimal controller for a Boeing 747 aircraft model [38], [39] under many different operating conditions. The linear systems are the same size ($N = 229$) but the condition numbers range from 50 to 2×10^{10}. The problems were solved using a 32-bit 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 3. As expected, double precision floating-point with 64 bits achieves better accuracy than the fixed-point implementations. However, single precision floating-point with 32-bits consistently achieves less accurate solutions. Since single precision only has 23 mantissa bits, a fixed-point implementation with 32 bits can provide better accuracy than a floating-point implementation with 32 bits if the problems are formulated such that the full dynamic range offered by a fixed representation can be efficiently utilized across different problems. Figure 3 also highlights that these problems are numerically challenging – if the scaling matrix (2) is not used, even the floating-point implementations fail to converge. This suggests that the proposed scaling procedure can also improve the numerical behaviour of floating-point iterative algorithms along with achieving its main goal for bounding variables. An example application supporting this claim is described in [39].

The final attainable accuracy $\|\hat{A}y - \hat{b}\|_2$, denoted by e_k, is determined by the machine unit round-off u_k. When using floor rounding $u_k := 2^{-k}$, where k denotes the number of bits used to represent the fractional part of numbers. It is well-known [40] that when solving systems of linear equations these quantities are related by

$$e_k \leq \mathcal{O}(\kappa(\hat{A})u_k)\,, \tag{17}$$

where $\kappa(\hat{A})$ is the condition number of the coefficient matrix. We have observed that, for $k \geq 15$, this relationship holds with approximate equality for all tested problems, including the problem described in Figure 2. For smaller bitwidths, excessive round-off error leads to unpredictable behaviour of the algorithm for the described application.

The constant of proportionality, which captures the numerical difficulty of the problem, is different for each problem. This constant cannot be computed *a priori*, but relationship (17) allows one to calculate it after a single simulation run and then determine the attainable accuracy when using a different number of bits, i.e. we can predict the numerical behaviour when using k bits by shifting the histogram for 32 fixed-point fraction bits.

5 Parameterizable FPGA Architecture

The results derived in Section 3 can be used to implement Lanczos-based algorithms reliably in low cost and low power fixed-point architectures, such as fixed-point DSPs and embedded microcontrollers. In this section, we will evaluate the potential efficiency improvements in FPGAs. In these platforms, for addition and subtraction operations, fixed-point units

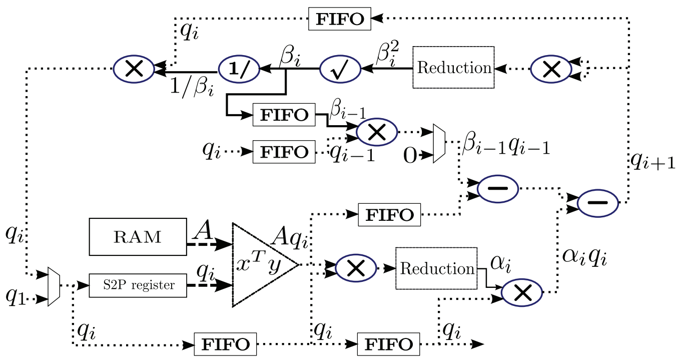


Fig. 4: Lanczos compute architecture. Dotted lines denote links carrying vectors, whereas solid lines denote links carrying scalars. The two thick dotted lines going into the x^Ty block denote N parallel vector links. The input to the circuit is q_1 going into the multiplexer and the matrix $\hat{A}$ being written into on-chip RAM. The output is α_i and β_i.

consume one order of magnitude fewer resources and incur one order of magnitude less arithmetic delay than floating-point units providing the same number of mantissa bits [41]. These platforms also provide flexibility for synthesizing different computing architectures. We now describe our architecture generating tool, which takes as inputs the data type, number of fraction bits k, level of parallelization P 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 described in VHDL.

The proposed compute architecture for implementing Algorithm 1 is shown in Figure 4. The most computationally intensive operation is the $\Theta(N^2)$ matrix-vector multiplication in Line 3 of Algorithm 1. This is implemented in the block labelled x^Ty, 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 of Algorithm 1 are all $\Theta(N)$ vector operations and $\Theta(1)$ scalar operations that use dedicated components.

The degree of parallelism in the circuit is parameterized by parameter P. For instance, if $P = 2$, there will be two x^Ty 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.

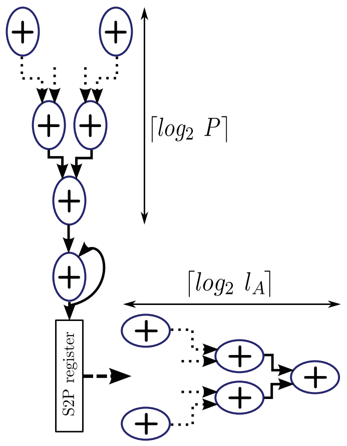


Fig. 5: Reduction circuit. Uses $P + l_A - 1$ adders and a serial-to-parallel shift register of length l_A.

TABLE 2: Delays for arithmetic cores. The delay of the fixed-point divider varies nonlinearly between 21 and 36 cycles from $k = 18$ to $k = 54$.

	l_A	l_M	l_D	l_{SQ}
fixed-point	1	2	-	$\lfloor \frac{k+1}{2} \rfloor + 1$
`float`	11	8	27	27
`double`	14	15	57	57

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

$$L := \left\lceil \frac{N}{P} \right\rceil + l_A \lceil \log_2 N \rceil + 5l_M + l_A + l_{SQ} + l_D + 2 + 2l_{red}, \tag{18}$$

where

$$l_{red} := \left\lceil \frac{N}{P} \right\rceil + l_A \lceil \log_2 P \rceil + l_A + l_A \lceil \log_2 l_A \rceil - 1 \tag{19}$$

is the number of cycles it takes the reduction circuit, illustrated in Figure 5, to reduce the incoming P streams to a single scalar value. This operation is necessary when computing $q_i^T A q_i$ and $r_{i+1}^T r_{i+1}$. Note in particular that for a fixed-point implementation with $l_A = 1$ and $P = 1$, the reduction circuit is a single adder and $l_{red} = N$, as expected. Table 2 shows the latency of the arithmetic units under different number representations.

For cases where N is large and the resources available cannot provide N parallel multipliers, it is possible to reduce the size of the dot-product unit, which consumes most resources. For the case where $P = \frac{1}{2}$, the dot product unit consists of $\lceil \frac{N}{2} \rceil$ parallel multipliers, followed by an adder tree of depth $\lceil \log_2 \lceil \frac{N}{2} \rceil \rceil$. An extra adder is needed at the output to add up the intermediate results, hence the overall latency of the dot-product computation does not change. The dotted links in Figure 4 would carry one value every two cycles.

5.1 Design Automation Tool

In order to evaluate the performance of our designs for a given target FPGA chip we created a design

TABLE 3: Resource usage

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

automation tool that generates optimum designs with respect to the following rule:

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

subject to

$$\mathbb{P}(e_k \leq \eta) > 1 - \xi \tag{20}$$

$$R(P,k) \leq \text{FPGA}_{\text{area}}, \tag{21}$$

where $L(P,k)$ is defined in (18) 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 to the desired accuracy for numerical reasons. The user can specify η – the tolerance on the error, and ξ – the proportion of problems allowed to fail to converge to the desired accuracy. In the remainder of the paper, we set $\xi = 10\%$, which is reasonable for the application domain for the data used. $R(P,k)$ is a vector representing the utilization of the different FPGA resources: flip-flops (FFs), look-up tables (LUTs), embedded multipliers and embedded RAM, for the Lanczos architecture illustrated in Figure 4 with parallelism degree P 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 (20) by making use of the information in Figure 3 and the relationship (17). Once k is fixed, find the maximum P such that (21) remains satisfied using the information in Table 3 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 [41]. Note that the actual resource utilization of the generated designs can differ slightly from the model predictions. However, the modeling error has been verified to be insignificant compared to the efficiency improvements that will be presented in Section 6.

Since the entire matrix has to be stored on-chip, memory is typically the limiting factor for implementations with a small number of bits. For larger numbers of bits embedded multipliers limit the degree of parallelization. In the former case, storage

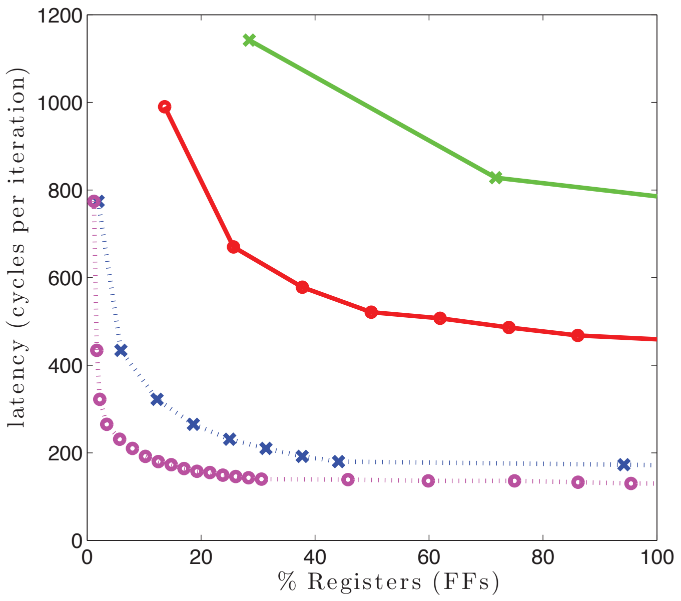


Fig. 6: Latency tradeoff against FF utilization (from model) on a Virtex 7 XT 1140 [42] for $N = 229$. Double precision ($\eta = 4.05 \times 10^{-14}$) and single precision ($\eta = 3.41 \times 10^{-7}$) are represented by solid lines with crosses and circles, respectively. Fixed-point implementations with $k = 54$ and 29 are represented by the dotted lines with crosses and circles, respectively. These Lanczos implementations, when embedded inside a MINRES solver, match the accuracy requirements of the floating-point implementations.

of some of the columns of $\hat{A}$ is implemented using banks of registers so FFs become the limiting resource. In the latter case, some multipliers are implemented using LUTs so these become the limiting resource. Figure 6 shows the trade-off between latency and FFs offered by the floating-point Lanczos implementations and two fixed-point implementations that, when embedded inside a MINRES solver, meet the same accuracy requirements as the single and double precision floating-point implementations. 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 while providing the same solution quality.

6 Performance Evaluation

In this section we will evaluate the relative performance of the fixed-point and floating-point implementations under the resource constraint framework of Section 5 for a Virtex 7 XT 1140 FPGA [42]. Then we will evaluate the absolute performance and efficiency of the fixed-point implementations against a high-end GPGPU with a peak floating-point performance of 1 TFLOP/s.

The trade-off between latency (18) and accuracy requirements for our FPGA implementations is investigated in Figure 7. For high accuracy requirements

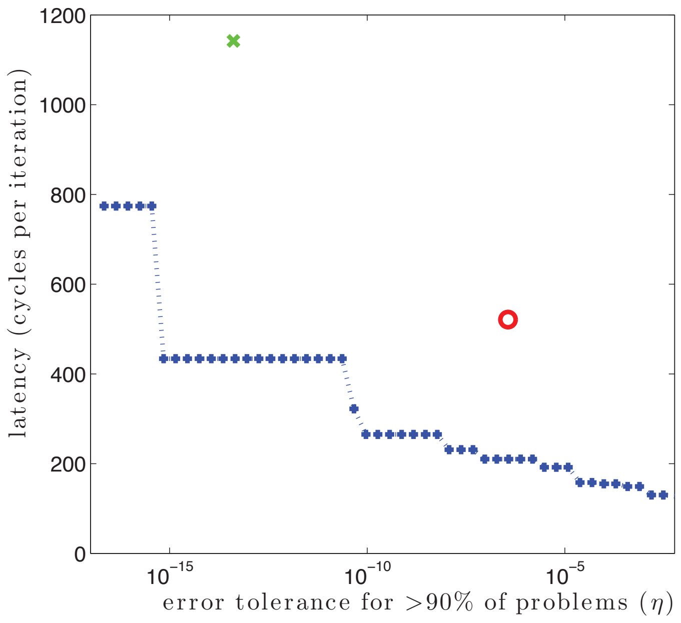
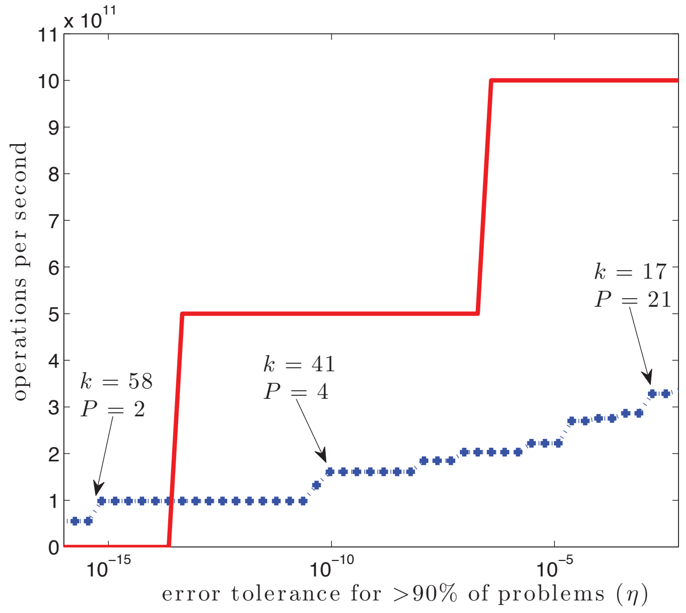


Fig. 7: Latency against accuracy requirements trade-off on a Virtex 7 XT 1140 [42] for $N = 229$. The dotted line, the cross and the circle represent fixed-point and double and single precision floating-point implementations, respectively. The jumps in the fixed-point curve mark the points at which the level of parallelization changes.

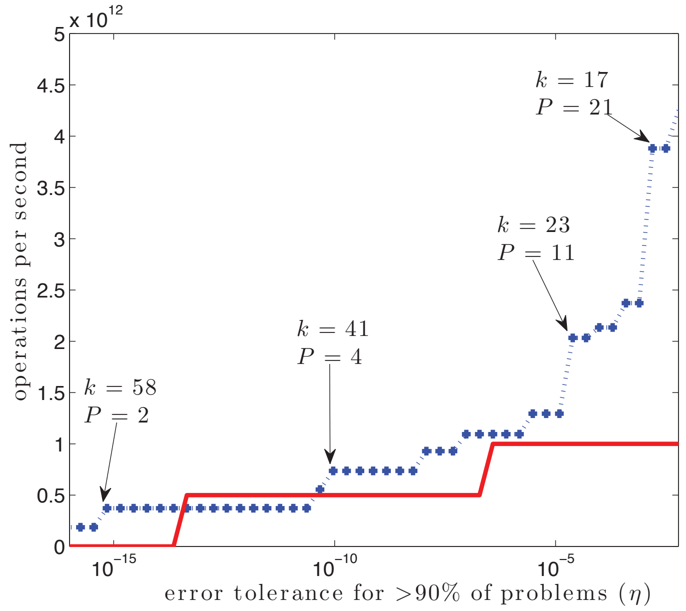
a large number of bits are needed reducing the extractable parallelism and increasing the latency. As the accuracy requirements are relaxed it becomes possible to reduce latency by increasing parallelism. The figure also shows that the fixed-point implementations provide a better trade-off even when 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 8. For extremely high accuracy requirements, not attainable by double precision floating-point, a fixed-point implementation can still achieve approximately 100 GOP/s. Since double precision floating-point only has 52 mantissa bits, a 53-bit fixed-point arithmetic can provide more accuracy if the dynamic range is controlled. For accuracy requirements of 10^{-6} and 10^{-3} the fixed-point implementations can achieve approximately 200 and 300 GOP/s, respectively. Larger problems would benefit more from incremental parallelization leading to greater performance improvements, especially for lower accuracy requirements.

The GPGPU curves are based on the NVIDIA C2050 [43], which has a peak single precision performance of 1.03 TFLOP/s and a peak double precision performance of 515 GFLOP/s. It should be emphasized that while the solid lines represent the



(a) $N = 229$, single problem



(b) $N = 229$, many problems (22)

Fig. 8: Sustained computing performance for fixed-point implementations on a Virtex 7 XT 1140 [42] for different accuracy requirements. The solid line represents the peak performance of a 1 TFLOP/s GPGPU. P and k are the degree of parallelization and number of fraction bits, respectively.

peak GPGPU performance, the actual sustained performance can differ significantly [44]. In fact, [45] reported sustained performance well below 10% of the peak performance when implementing the Lanczos kernel on this GPGPU.

The trade-off between performance and accuracy requirements is important for the range of applications that we consider. For some HPC applications, high accuracy requirements, even beyond double precision, can be a high priority. On the other hand, for some embedded applications that require the repeated solution of similar problems, accuracy can be sacrificed for the ability to apply actions fast and respond quickly to new events. In some of these applications, solution accuracy requirements of 10^{-3} can be perfectly reasonable.

The results presented so far have assumed that we are processing a single Lanczos problem at a time. Using this approach the arithmetic units in our circuit are always idle for some fraction of the iteration time. In addition, because the constant term in (18) is relatively large, the effect of incremental parallelization on latency reduction becomes small very quickly. In the situation when there are many independent problems available [46], it is possible to use the idle computational power by time-multiplexing multiple problems into the same circuit to hide the pipeline latency and keep arithmetic units busy [47]. The number of problems needed to fill the pipeline is given by the following expression

$$\left\lceil \frac{L \text{ from } (18)}{\lceil \frac{N}{P} \rceil} \right\rceil . \tag{22}$$

If the extra storage needed does not hinder the achievable parallelism, 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 8 (b). Using this approach there is a more direct transfer between parallelization and sustained performance. The sharp improvement in performance for low accuracy requirements is a consequence of a significant reduction in the number of embedded multiplier blocks necessary for implementing multiplier units, allowing for a significant increase in the available resources for parallelization.

For the Virtex 7 XT 1140 [42] FPGA from the performance-optimized Xilinx device family, Xilinx power estimator [48] was used to estimate the maximum power consumption at approximately 22 Watts. For the C2050 GPGPU [43], the power consumption is in the region of 100 Watts, while a host processor consuming extra power would still be needed for controlling the data transfer to and from the GPGPU. Hence, for problems with modest accuracy requirements, there will be more than one order of magnitude difference in power efficiency when measured in terms of operations per watt between the sustained fixed-point FPGA performance and the peak GPGPU floating-point performance.

7 Other Linear Algebra Kernels

It is expected that the same scaling procedure presented in Section 3 will also be useful for bounding variables in other iterative linear algebra algorithms based on matrix-vector multiplication.

Algorithm 3 Arnoldi algorithm

```
Require: Initial iterate $q_1$ such that $\|q_1\|_2 = 1$ and
    $h_{1,0} := 1$.
 1: for $i = 1$ to $i_{max}$ do
 2:   $q_i \leftarrow \frac{r_{i-1}}{h_{i,i-1}}$
 3:   $z \leftarrow Aq_i$
 4:   $r_i \leftarrow z$
 5:   for $k = 1$ to $i$ do
 6:     $h_{k,i} \leftarrow q_k^T z$
 7:     $r_i \leftarrow r_i - h_{k,i} q_k$
 8:   end for
 9:   $h_{i+1,i} \leftarrow \|r_i\|_2$
10: end for
11: return $h$
```

The standard Arnoldi iteration [13], described in Algorithm 3, transforms a non-symmetric matrix $A \in \mathbf{R}^{N \times N}$ into an upper Hessenberg matrix H (upper triangle and first lower diagonal are non-zero) with similar spectral properties as A using an orthogonal transformation matrix Q. At every iteration the approximation is refined such that

$$Q_i^T A Q_i = H_i =: \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & \cdots & h_{1,k} \\ h_{2,1} & h_{2,2} & & & \vdots \\ 0 & h_{3,2} & \ddots & & \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & & & h_{k,k-1} & h_{k,k} \end{bmatrix},$$

where $Q_i \in \mathbf{R}^{N \times i}$ and $H_i \in \mathbf{R}^{i \times i}$.

Since the matrix A is not symmetric it is not necessary to apply a symmetric scaling procedure; hence, instead of solving $Ax = b$, we solve

$$M^2 Ax = M^2 b$$
$$\Leftrightarrow \hat{A}x = \hat{b}$$

and the computed solution remains the same as the solution to the original problem. The following proposition summarises the variable bounds for the Arnoldi process:

Proposition 2. *Given the scaling matrix (2), the Arnoldi iteration applied to $\hat{A}$, for any non-singular matrix A, has intermediate variables with the following bounds for all i, j and k:*

- $[q_i]_k \in [-1, 1]$
- $[\hat{A}]_{kj} \in [-1, 1]$
- $[\hat{A}q_i]_k \in [-1, 1]$
- $[H]_{kj} \in [-1, 1]$

where i denotes the iteration number and $[]_k$ and $[]_{kj}$ denote the k^{th} component of a vector and kj^{th} component of a matrix, respectively.

Proof: According to the proof of Lemma 1, the spectral radius of the non-symmetric scaled matrix is still bounded by $\rho(\hat{A}) \leq 1$. As with the Lanczos iteration, the eigenvalues of the approximate matrix H_i are contained within the eigenvalues of $\hat{A}$ even throughout the intermediate iterations. One can use the relationship (8) to show that the coefficients of the Hessenberg matrix are bounded by $\rho(\hat{A})$. The bounds for the remaining expressions in the Arnoldi iteration are obtained in the same way as in Theorem 1. □

It is expected that the same techniques could be applied to other related kernels such as the unsymmetric Lanczos process or the power iteration for computing maximal eigenvalues.

8 Conclusion

Fixed-point computation is more efficient than floating-point from the digital circuit point of view. We have shown that fixed-point computation can also be suitable for problems that have traditionally been considered floating-point problems if enough care is taken to formulate these problems in a numerically favourable way. Even for algorithms known to be vulnerable to numerical round-off errors, accuracy does not necessarily have to be compromised by moving to fixed-point arithmetic if the dynamic range can be controlled such that a fixed-point representation can represent numbers efficiently.

Implementing an algorithm using fixed-point arithmetic gives more responsibility to the designer, since all variables need to be bounded in order to avoid overflow errors that can lead to unpredictable behaviour. We have proposed a scaling procedure that allows us to bound and control the dynamic range of all variables in the Lanczos method – the building block in iterative methods for solving the most important linear algebra problems, which are ubiquitous in engineering and science. The proposed methodology is simple to implement but uses linear algebra theorems to establish bounds, which is currently well beyond the capabilities of state-of-the-art automatic tools for solving the bounding problem.

The capability for implementing these algorithms using fixed-point arithmetic could have an impact both in the high performance and embedded computing domains. In the embedded domain, it has the potential to open up opportunities for implementing sophisticated functionality in low cost systems with limited computational capabilities. For high-performance scientific applications it could help in the effort to reach exascale levels of performance while keeping the power consumption costs at an affordable level. For other applications there are substantial processing performance and efficiency gains to be realized.

Acknowledgments

The authors would like to acknowledge the support of the EPSRC (Grants EP/G031576/1 and EP/I012036/1) and the EU FP7 Project EMBOCON, as well as industrial support from Xilinx, the Mathworks, and the European Space Agency. The authors would also like to acknowledge fruitful discussions with Mr Theo Drane and Dr Ammar Hasan, and would like to thank Dr Ed Hartley for providing the benchmark set of initial conditions for the Boeing 747 optimal controller.

References

[1] 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.

[2] J. Demmel, *Applied Numerical Linear Algebra*, 1st ed. Society for Industrial and Applied Mathematics, 1997.

[3] 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.

[4] J. M. Maciejowski, *Predictive Control with Constraints*. Harlow, UK: Pearson Education, 2001.

[5] S. Hemmert, "Green HPC: From nice to necessity," *Computing in Science and Engineering*, vol. 12, no. 6, pp. 8–10, Nov 2010.

[6] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed. Morgan Kaufmann Publishers, 2011.

[7] D. Jensen and A. Rodrigues, "Embedded systems and exascale computing," *Computing in Science & Engineering*, vol. 12, no. 6, pp. 20–29, 2010.

[8] C. C. Paige, "Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix," *Journal of the Institute of Mathematics and Applications*, vol. 18, pp. 341–349, 1976.

[9] C. Inacio, "The DSP decision: fixed point or floating?" *IEEE Spectrum*, vol. 33, no. 9, pp. 72–74, 1996.

[10] 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.

[11] J. L. Jerez, G. A. Constantinides, and E. C. Kerrigan, "Fixed-point Lanczos: Sustaining TFLOP-equivalent performance in FPGAs for scientific computing," in *Proc. 20th IEEE Symp. on Field-Programmable Custom Computing Machines*, Toronto, Canada, Apr 2012, pp. 53–60.

[12] M. Daumas and G. Melquiond, "Certification of bounds on expressions involving rounded operators," *Transactions on Mathematical Software*, vol. 37, no. 1, 2010.

[13] W. E. Arnoldi, "The principle of minimized iterations in the solution of the matrix eigenvalue problem," *Quarterly in Applied Mathematics*, vol. 9, no. 1, pp. 17–29, 1951.

[14] L. Zhuo and V. K. Prasanna, "High-performance designs for linear algebra operations on reconfigurable hardware," *IEEE Transactions on Computers*, vol. 57, no. 8, pp. 1057–1071, 2008.

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

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

[17] 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 *Proc. 24th ACM Int. Conf. on Supercomputing*, Tsukuba, Japan, Jun 2010, pp. 325–335.

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

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

[24] G. A. Constantinides, "Tutorial paper: Parallel architectures for model predictive control," in *Proc. European Control Conf.*, Budapest, Hungary, Aug 2009, pp. 138–143.

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

[26] R. E. Moore, *Interval Analysis*. Englewood Cliff, NJ, USA: Prentice-Hall, 1966.

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

[28] D. Boland and G. A. Constantinides, "A scalable approach for automated precision analysis," in *Proc. ACM Symp. on Field Programmable Gate Arrays*, Monterey, CA, USA, Mar 2012, pp. 185–194.

[29] L. D. Moura and N. B. Bjørner, "Z3: An efficient SMT solver," in *Proc. Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, Budapest, Hungary, Mar 2008, pp. 337–340.

[30] P. S. Chang and A. N. Willson, "Analysis of conjugate gradient algorithms for adaptive filtering," *IEEE Transactions on Signal Processing*, vol. 48, no. 2, pp. 409–418, 2000.

[31] R. I. Hernández, R. Baghaie, and K. Kettunen, "Implementation of Gram-Schmidt conjugate direction and conjugate gradient algorithms," in *Proc. IEEE Finish Signal Processing Symp.*, Oulu, Finland, May 1999, pp. 165–169.

[32] P. Jdnis, M. Melvasalo, and V. Koivunen, "Fast reduced rank equalizer for HSDPA systems based on Lanczos algorithm," in *Proc. IEEE 7th Workshop on Signal Processing Advances in Wireless Communications*, Cannes, France, Jul 2006, pp. 1–5.

[33] J. L. Lions, "ARIANE 5 Flight 501 Failure," Report by the Inquiry Board, Paris, France, http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html, Jul 1996.

[34] J.-M. Muller, *Elementary Functions: Algorithms and Implementation*, 2nd, Ed. Birkhaeuser, 2006.

[35] J. H. Wilkinson, *Rounding Errors in Algebraic Processes*, 1st ed., ser. Notes on Applied Science. London, UK: Her Majesty's Stationary Office, 1963, no. 32.

[36] C. C. Paige, "Accuracy and effectiveness of the Lanczos algorithm for the symmetric eigenproblem," *Linear Algebra and its Applications*, vol. 34, pp. 235–258, Dec 1980.

[37] 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

[38] 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. http://www.faulttolerantcontrol.nl/. [Online]. Available: http://www.faulttolerantcontrol.nl/

[39] E. Hartley, J. L. Jerez, A. Suardi, J. M. Maciejowski, E. C. Kerrigan, and G. A. Constantinides, "Predictive control using an FPGA with application to aircraft control," *IEEE Transactions on Control Systems Technology*, 2013, (accepted).

[40] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. SIAM, 2002.

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

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

[43] NVIDIA. (2013, May) Tesla C2050 GPU computing processor. NVIDIA. [Online]. Available: http://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_C2050_C2070_jul10_lores.pdf

[44] K. Fatahalian, J. Sugerman, and P. Hanrahan, "Understanding the efficiency of GPU algorithms for matrix-matrix multiplication," in *Proc. ACM Conf. on Graphics Hardware*, Grenoble, France, Aug 2004, pp. 133–137.

[45] A. Rafique, N. Kapre, and G. A. Constantinides, "A high throughput FPGA-based implementation of the lanczos method for the symmetric extremal eigenvalue problem," in *Proc. Int. Symp. on Appied Reconfigurable Computing*, 2012, pp. 239—250.

[46] 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*, vol. 6, no. 8, pp. 1029–1041, 2012.

[47] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1-6, pp. 5–35, 1991.

[48] Xilinx. (2012, Dec) Xilinx power estimator. Xilinx. [Online]. Available: http://www.xilinx.com/ise/power_tools/license_7series.htm

PLACE PHOTO HERE

Juan Luis Jerez (S'11) received the M.Eng degree in electrical engineering from Imperial College London, UK in 2009 and a Ph.D from the Circuits and Systems research group at the same institution in 2013. He is currently a postdoctoral researcher at ETH Zürich. The focus of his research work is on developing tailored linear algebra and optimization algorithms for efficient implementation on custom parallel computing platforms and embedded systems.

PLACE PHOTO HERE

George A. Constantinides (S'96-M'01-SM'08) received the Ph.D. degree from Imperial College London in 2001. Since 2002, he has been with the faculty at Imperial College London, where he is currently Professor of Digital Computation and Head of the Circuits and Systems research group. He will be program (general) chair of the ACM International Symposium on Field-Programmable Gate Arrays in 2014 (2015). He serves on several programme committees and has published over 150 research papers in peer refereed journals and international conferences. Dr Constantinides is a Senior Member of the IEEE and a Fellow of the British Computer Society.

PLACE PHOTO HERE

Eric C. Kerrigan (S'94-M'02) received a PhD from the University of Cambridge in 2001 and has been with Imperial College London since 2006. His research is focused on the development of efficient numerical methods and embedded computing architectures for solving advanced optimization, control and estimation problems in real-time. His work is applied to a variety of problems in aerospace and renewable energy applications. He is on the IEEE Control Systems Society Conference Editorial Board and is an associate editor of the IEEE Transactions on Control Systems Technology, Control Engineering Practice, and Optimal Control Applications and Methods.