

MORE FLOPS OR MORE PRECISION? ACCURACY PARAMETERIZABLE LINEAR EQUATION SOLVERS FOR MODEL PREDICTIVE CONTROL *

Antonio Roldao Lopes[◊], Amir Shahzad[◊], George A. Constantinides[◊], Eric C. Kerrigan^{◊,†}

[◊]Department of Electrical and Electronic Engineering, [†] Department of Aeronautics
Imperial College London
{aroldao,a.shahzad07,gac1,e.kerrigan}@imperial.ac.uk

ABSTRACT

In this paper we exploit FPGA flexibility in the context of accelerating the solution of many small systems of linear equations, a problem central to model predictive control (MPC). The main observation exploited by this work is the distinction between accuracy (meaning the degree of correctness of a final computational result) and precision (meaning the degree of correctness of each atomic computation) [1]. Using iterative methods for solving linear systems, one can obtain improved accuracy either by running more iterations or by using more precise internal computations, unlike direct methods, where accuracy is only a function of operation precision. Thus, in iterative methods, for a given accuracy requirement we may conduct fewer iterations in a higher precision, or more in a lower precision. We argue that this suits FPGA architectures ideally, as low precision operations result in greater parallelism for any fixed area constraint. We show that we may therefore optimize the performance by balancing iteration count and operation precision, resulting in a several-fold speed improvement over a double-precision implementation, but with the same final result accuracy. Exploring this trade-off it is possible to provide a speed-up of $26\times$ on average, $14\times$ in the worst case and $36\times$ in the best, compared to a high-end CPU running at 3.0 GHz. This has the potential to allow modern high-performance control techniques to be used in novel settings such as aircraft and diesel engines.

1 Introduction

Field Programmable Gate Arrays have reached density levels beyond a million equivalent logic gates and started to be embedded in highly optimized hardware units such as memory blocks, high-speed transceivers, arithmetic operators, or complete micro-processors [2]. These architectures

provide an unprecedented degree of freedom by combining the flexibility supplied by the configurable blocks and routing with the optimized hardware units. When this flexibility is used to interconnect and instantiate a highly efficient data-path, FPGAs can significantly out-perform high-end CPUs. Reported applications that have demonstrated high performance of FPGAs are varied and include scientific computing [3], medical imaging [4], robotics [5], financial systems [6], and security encryption [7].

In this paper we examine FPGA flexibility in the context of accelerating multiple solutions of small systems of linear equations, with application to Model Predictive Control. Model Predictive Control (MPC) is a method used for controlling multi-variable dynamical systems. This method has become an industry standard (mainly in the petrochemical industry) due to its intrinsic capability for dealing with hard constraints [8]. The main idea behind MPC is to choose the control action by repeatedly solving an optimal control problem over a certain horizon. From this solution, only the first action is implemented. This action can be constrained to a defined range representing real physical limitations. A new output sample is then measured and the process is repeated. As illustrated in Fig. 1, this control system requires an internal system model of the process from which it predicts the future outputs, a history of past control moves, and an optimization cost function over the prediction horizon [8]. At the heart of this method there is a computationally intensive optimization problem. The complexity of this problem depends on factors that include prediction time horizon length, as well as the number of input and output variables and constraints. Because solving this optimization problem requires a high computational effort, MPC has been restricted to slow processes such as chemical plants. However, with the advances in FPGAs and their speed-up through the methods described in this paper, it is possible to push this boundary further and apply MPC to faster real-time systems.

The main contribution of this paper is centred on the trade-offs between iteration count, computational precision,

*THE AUTHORS WOULD LIKE TO ACKNOWLEDGE THE SUPPORT OF THE EPSRC (GRANT EP/C549481/1 AND EP/E00024X/1)

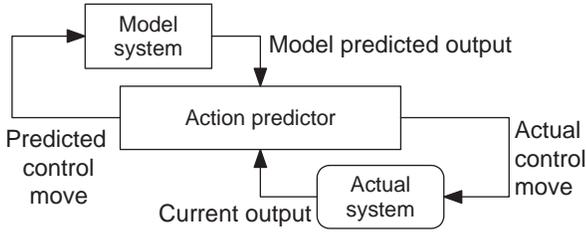


Fig. 1. Model Predictive Control Overview.

hardware utilization, and execution time in order to maximize the speed-up of solution finding for many small linear systems. The possibility to combine these trade-offs arises in iterative methods due to the additional possibility of controlling the operating time through the number of iterations executed; in direct methods, such as LU or Cholesky decompositions, the accuracy of the solution is only dependent on number system precision used.

In Section 2 we give a description of related work and short description of Model Predictive Control and how the necessity to solve systems of linear equations arises. This section is followed by the description of a Conjugate Gradient solver. Section 5 describes the trade-offs between precision and parallelism to provide increased speed-up compared to IEEE standard floating point, but with the same final accuracy. Section 6 concludes the paper.

2 Related Work and Background

2.1 Related Work

Previous work can be divided into two categories. One set of work aims to maximize raw floating point performance (FLOPS) in an IEEE standard representation, whereas the other is focused on mixing different precision operations to find a solution within a specified accuracy.

In the first category, acceleration is mainly provided by exploiting wide-parallelism, deep-pipelining, and efficient data-paths. Examples of such works include implementations of both direct methods and iterative methods for solution finding of systems of linear equations.

Important work in direct methods includes implementations of Cholesky and LU decompositions. The LU decomposition work demonstrated that the FPGA can outperform a single processor by $2.2\times$ and that the energy dissipated per computation is a factor $5\times$ less [9]. The Cholesky implementation demonstrated a performance increase of $2\times$ over software, for matrices of order 48, on a APEX EP20K1500E FPGA [10]. This implementation was based around a system that uses an asymmetric, shared-memory MIMD architecture and was built using two embedded Nios processors.

Relevant work in iterative methods includes implementations of a Jacobi method [11], a Conjugate Gradient method [12], and a MINRES method [13]. The Jacobi solver was implemented on a Xilinx VirtexII Pro XC2VP50 where performance estimates, which include both data transfer and execution time, show that this circuit provides a $1.3\times$ speedup with a large dense matrix, for a single iteration, when compared to a uniprocessor implementation. For a single iteration and a large sparse matrix, this Jacobi circuit could achieve an estimated acceleration of $1.1\times$ to $19.5\times$. Multiple iteration speedups ranged from $2.8\times$ to $36.8\times$. The CG work described a deeply-pipelined and widely-parallel Conjugate Gradient implementation using the IEEE 754 single precision floating point standard for problems with a banded input matrix. It was demonstrated that matrix bands up to 92 are achievable, with peak floating point performance above 32 GFLOPS for the Virtex5-330T device. The minimum residual algorithm (MINRES) work examined an IEEE single precision floating point implementation of the MINRES algorithm. It demonstrated that through parallelisation and heavy pipelining of all floating point components it is possible to achieve a sustained performance of up to 53 GFLOPS on the Virtex5-330T.

In the second category, acceleration is provided by exploiting mixed-precision operations. The key idea behind these mixed-precision schemes is to perform as many operations as possible in lower precision, and only execute a number of crucial operations in the slower high-precision [14]. Studies have shown that these schemes can obtain exactly the same solution accuracy as if the entire computation was performed in high-precision [15]. Junqing et al. have implemented an FPGA-based mixed-precision solver using the Cray-XD1 and concluded that it is possible to achieve two to three times better performance than a double-precision design running on a CPU [16].

In contrast, this work uses a single operating precision, but explicitly tunes this precision to optimize performance: too low and the algorithm will take too many iterations to converge, too high and the parallelism available will be reduced. The possibility to exploit this trade-off between precision and iteration count with parallelism arises only in iterative methods. This work exploits this trade-off technique in the context of accelerating Model Predictive Control.

2.2 Model Predictive Control

The MPC problem for a linear discrete-time state-space system is a finite horizon optimal control problem, which can be defined as the minimization of

$$V(x_0, u_0, x_1, u_1, \dots, u_{N-1}, x_N) := x_N^T P x_N + \sum_{i=0}^{N-1} (x_i^T Q x_i + u_i^T R u_i + 2x_i^T M u_i) \quad (1a)$$

subject to equality constraints:

$$x_0 = x \quad (1b)$$

$$x_{i+1} = Ax_i + Bu_i, \quad i = 0, 1, \dots, N-1 \quad (1c)$$

and inequality constraints

$$J_i x_i + E_i u_i \leq d_i, \quad i = 0, 1, \dots, N-1 \quad (1d)$$

$$J_N x_N \leq d_N \quad (1e)$$

where N is the horizon length and $x \in \mathbf{R}^n$ is the current state. (1a) describes the quadratic objective function [8]. (1c) represents the model of the system dynamics. The matrices A, B are assumed time-invariant, and $u_i \in \mathbf{R}^m$ is the input vector. (1d,1e) represent the inequality constraints on state and input vectors. The terminal weighting matrix P is calculated off-line [8].

The solution of the problem gives the optimal input sequence $\{u_0^*(x), u_1^*(x), \dots, u_{N-1}^*(x)\}$, which minimises the objective function (1a) subject to given constraints. The first input $u_0^*(x)$ is applied to the system to be controlled and this procedure is repeated at each successive control interval. The problem (1) is easily converted to a strictly convex quadratic programming (QP) problem of the form [8] :

$$\min_{\mathbf{u}} \frac{1}{2} \mathbf{u}^T H \mathbf{u} + \mathbf{u}^T D x \quad \text{s.t.} \quad G \mathbf{u} \leq g + F x \quad (2)$$

where H is a symmetric positive definite matrix of dimension mN . The matrices D, G, F and vector g are of appropriate dimensions and $\mathbf{u} := [u_0^T \dots u_{N-1}^T]^T$ is the unknown vector.

Solving the QP problem on-line has restricted its application to slow systems, where sample times are in seconds or minutes [17]. However, due to rapid growth in processing power, recent attempts have been made to show MPC's efficacy in controlling fast processes such as fighter jet aircrafts and diesel engines [18, 19].

In this paper, we have used Logarithmic Barrier Interior Point method (LBIPM) to solve the QP problem. In this method we minimize a composite function that reflects the original objective function as well as the influence of constraints. The LBIPM converts the inequality constrained problem into a sequence of equality constrained problems [20]. This sequence of problems are then solved using Newton's method.

2.3 Newton's Method

This is an important iterative method for solving a system of nonlinear equations. It requires the first order $\nabla \mathcal{L}(\mathbf{u})$ and the second order derivative $\nabla^2 \mathcal{L}(\mathbf{u})$ of the associated Lagrangian of the LBIPM and an initial guess of the unknown \mathbf{u} . The complete algorithm is described in Algorithm 1. The p_i is the search direction and α is the step

length which can be calculated using backtracking line search algorithm described in Section 9.2 of [20]. The main step which involves most of the computation in this method is the solution of a linear system, which can be solved using an iterative method.

Algorithm 1 Newton's Method

Given:

$\nabla^2 \mathcal{L}(\mathbf{u}), \nabla \mathcal{L}(\mathbf{u})$, initial guess \mathbf{u}_0 and tolerance ϵ .

Output:

The optimal \mathbf{u} .

Algorithm:

- 1: Set $i = 0$
 - 2: **while** $\|\nabla \mathcal{L}(\mathbf{u}_i)\| < \epsilon$ **do**
 - 3: Solve $\nabla^2 \mathcal{L}(\mathbf{u}_i) p_i = -\nabla \mathcal{L}(\mathbf{u}_i)$.
 - 4: $\mathbf{u}_i = \mathbf{u}_i + \alpha p_i$, where α is calculated by a line search algorithm.
 - 5: $i = i + 1$
 - 6: **end while**
-

For simplicity, the linear system appearing at line 3 of Algorithm 1 will be denoted as $\mathbf{A} \mathbf{x} = b$ in following sections. To solve this linear system there are a number of methods that have been well-studied. The following sections explore a particular iterative method and how it can be optimized to find a sufficiently accurate solution.

2.4 Solving Systems of Linear Equations

There are two classes of solvers of systems of linear equations in the form described in (3): direct methods, where a block of computation is performed and the solution given at the end, and iterative methods, where a relatively smaller computation block is repeatedly performed and at each iteration the solution refined. In both classes there a number of algorithms that exploit the properties of matrix A in order to reduce computational time. These properties can be related to structure, such as the matrix \mathbf{A} being symmetric (*i.e.* $\mathbf{A} = \mathbf{A}^T$), or other attributes such as being positive definite (*i.e.* $\mathbf{x}^T \mathbf{A} \mathbf{x} > 0$ for all non-zero real vectors \mathbf{x}). Two general cases for solution finding of systems of linear equations include Gaussian Elimination (direct) and the Generalized Minimal Residual method (iterative). Two methods that apply to the particular matrix type arising in our problem are the Cholesky decomposition (direct) and the Conjugate Gradient (CG) method (iterative).

$$\mathbf{Ax} = b$$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \quad (3)$$

Iterative methods for solving systems of linear equations possess a number of crucial differences compared to direct methods. One particular difference, which is being explored in this paper, is the ability to trade-off execution iteration count for individual computational precision for the same solution accuracy. The trade-offs between these factors is not possible in direct methods since the solution accuracy is a consequence of the individual computations precisions alone. Another important aspect that makes iterative methods suitable for application in FPGAs, is their heavy reliance on matrix-by-vector multiplications. These operations are highly parallelisable and comprise only of multipliers and adders that map very efficiently in hardware [21].

2.5 Conjugate Gradient Method

The algorithm detailed in Algorithm 2, consists of two parts. The first is an initialization that produces a ‘residual’ or search direction. The second part iterates until the relative residual error is acceptable.

Algorithm 2 Conjugate Gradient (CG) method

Given:

\mathbf{A} , b , initial guess of \mathbf{x} and error tolerance ϵ

Output:

Solution of linear system \mathbf{x} such that $\|\mathbf{Ax} - b\|_2 \leq \epsilon \|b\|_2$

Algorithm:

- 1: $r = b - \mathbf{Ax}$
 - 2: $d = r$
 - 3: $\delta_{new} = r^T r$
 - 4: $\epsilon_f = \epsilon^2 b^T b$
 - 5: **repeat**
 - 6: $q = \mathbf{A}d$
 - 7: $\alpha = \delta / d^T q$
 - 8: $\mathbf{x} = \mathbf{x} + \alpha d$
 - 9: $r = r - \alpha q$
 - 10: $\delta_{old} = \delta_{new}$
 - 11: $\delta_{new} = r^T r$
 - 12: $\beta = \delta_{new} / \delta_{old}$
 - 13: $d = r + \beta d$
 - 14: **until** ($\delta_{new} < \epsilon_f$)
-

Under infinite precision, CG provides the solution \mathbf{x} of $\mathbf{Ax} = b$ in at most n iterations, where n is the order of

matrix \mathbf{A} . However, under finite precision the number of required iterations is potentially unbounded. Hence, it is important to implement the CG algorithm with enough computational precision to convergence to the intended solution, in practice.

2.6 Floating Point in Hardware

For the implementation described in the next section, we have used Xilinx Core Generator Floating Point Modules version 3 [22]. These modules can be fine-tuned to implement a variety of exponents, mantissas, or latencies. In our design, since we are targeting multiple independent computations, to maximize throughput, all the modules have the highest possible latency.

In Fig. 2 we have plotted how the mantissa width affects the operational frequency for the various floating point cores. In this plot, it is possible to observe that the weakest clock frequency is likely to be determined by the adder frequency for low mantissas and the divider frequency for high mantissas.

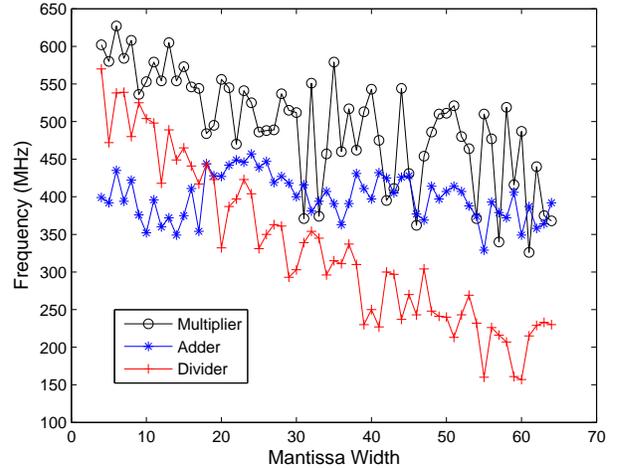


Fig. 2. Floating point core operating frequency vs mantissa width.

3 Parallel Hardware Conjugate Gradient with Parameterizable Word-length

For this work we adapted an existing Conjugate Gradient implementation [21] to allow the incorporation of a multitude of different word-length Floating Point modules. This design implements a fully pipelined CG iteration, with the matrix-vector operation $\mathbf{A}d$ being processed in a block of n steps, as described in Fig. 5. This compromise between a block-based matrix-vector and a fully-parallelized version,

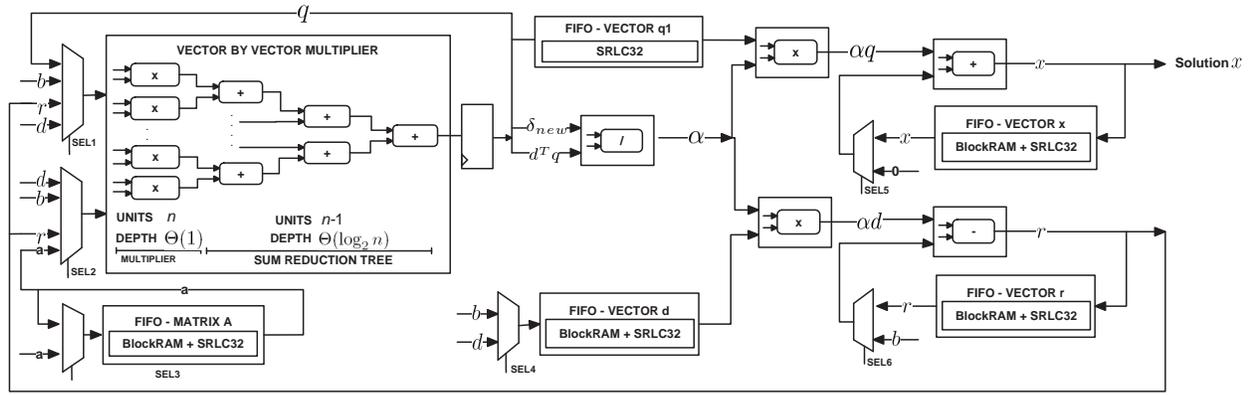


Fig. 3. Partial circuit schematic displaying the vector-by-vector module, two of constant by vector multiplications, a vector by vector summation, vector by vector subtraction and FIFOs. Some of these FIFOs use a combination of Xilinx SRLC32 primitives and BlockRAMs and store various vectors including A matrices in the row-by-row form.

provides significant resource savings and allows the design to have scalable I/O requirements. This previous design, as partially depicted in Fig. 3, already allowed the customization of matrix order, n , FP latencies, L_m , L_s , L_d (for the multiplier, adder and divider respectively), and was easily adapted for our experiment. This experiment investigated effects of a range (from 4 to 55 bits) of FP mantissa widths on the operating frequency and resource utilization of the CG circuit. These results were gathered post placement-and-routing using Xilinx’s ISE version 10.1.

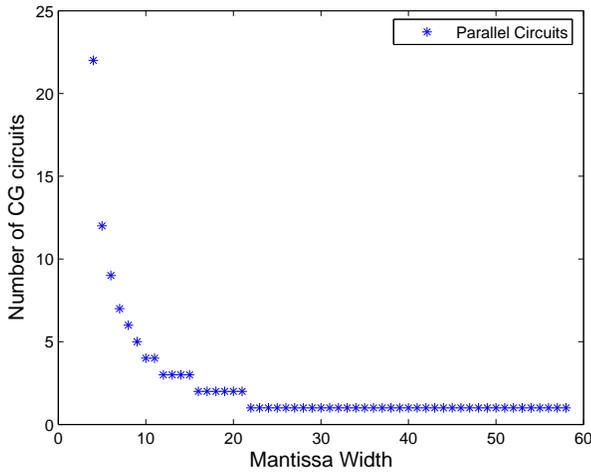


Fig. 4. Number of CG circuits embeddable in a Virtex5-330T as function of FP mantissa width.

Fig. 4 plots the number of parallel copies of the data-path in Fig. 5 implementable on a high-capacity FPGA. This number reached a plateau of one circuit for mantissas above 21-bits and increases up to 22 circuits for the lowest mantissa-width of 4-bits.

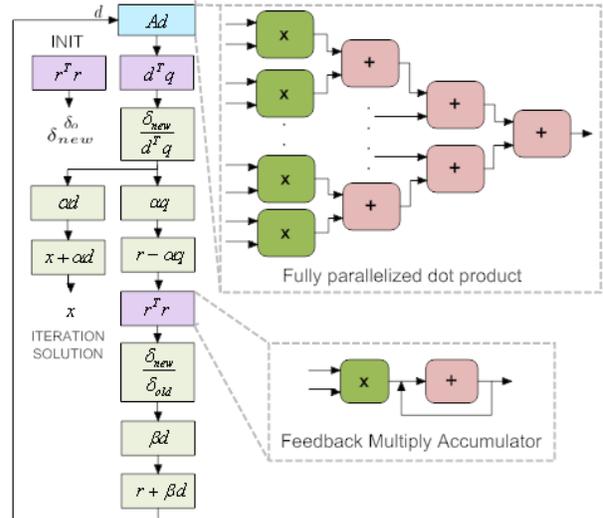


Fig. 5. Conjugate Gradient circular data-path showing a row-by-row-based matrix-by-vector multiplication and a Multiply Accumulator-based vector-by-vector multiplier.

Fig. 6 illustrates overall iteration performance by the varying latencies per iteration introduced by deeper floating point units (Fig. 7), the clock rate (Fig. 2) and the number of circuits (Fig. 4).

Fig. 6 shows a steady increase in the time per iteration, due to reduced clock rate, combined with sharp jumps due to decreasing opportunity for parallelism. However for the same accuracy, reduced precision can cause a (data-dependent) increase in iteration count, counteracting this effect. Thus it is essential to test this impact on real data, requiring the development of an MPC test-bench.

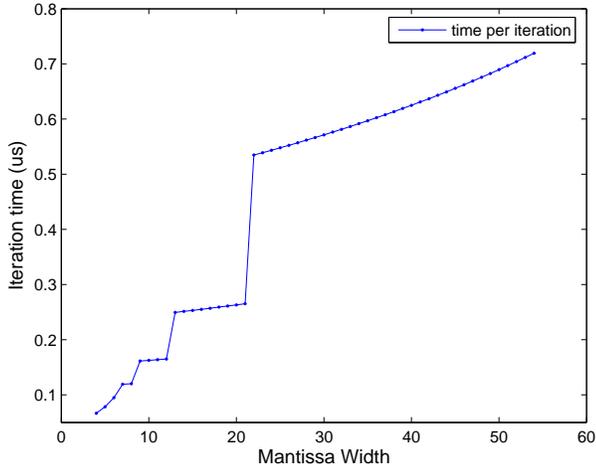


Fig. 6. Computational slow-down with increased mantissa width.

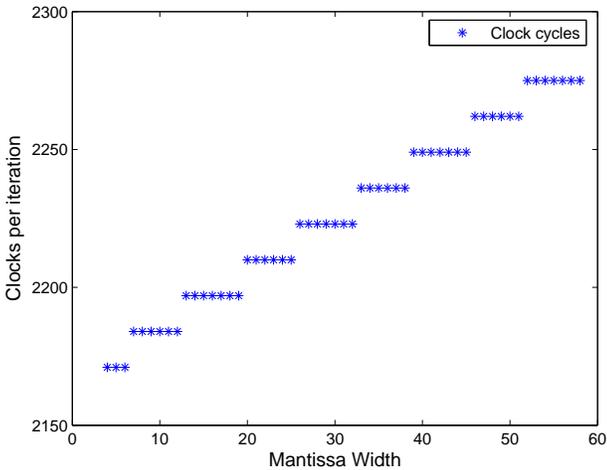


Fig. 7. Number of clocks per iteration vs mantissa width.

4 MPC testbench

For this work we have used a classic example from the MPC literature that involves controlling the model of a Cessna Citation 500 aircraft to a certain altitude [8].

The system’s state vector x constitutes angle of attack, pitch angle, pitch rate and altitude, the output vector y represents pitch angle, altitude and altitude rate and the controlling input u is elevator angle. The continuous state-space model is converted into discrete state-space model with sample time of 0.1 sec. The input is constrained to $|u| \leq 0.262$ (15°) and the pitch angle to 20° for passenger comfort. With this scenario as the MPC test-case, Matlab is used to solve the constrained MPC problem with logarithmic barrier interior point method. The various runs of the simulation are carried out with randomly perturbed initial conditions. These runs produced 3290 different linear systems problems in the form described in the Newton’s step in Section 2.3. These linear systems are all symmetric, positive-definite and their condition numbers range between 1370 and 431983. This data was subsequently used to empirically analyze the precision and mantissa requirements, as described in the next section.

4.1 Optimizing the number system

Using the problems generated in the previous section, we needed to know how the floating point mantissa width affected the solution in our adapted Conjugate Gradient hardware. For this we have encoded an ANSI-C representation of the hardware CG and used MPFR (multiple-precision floating-point) library [23] to emulate various precisions/mantissa widths FP operations. In order for this software to be truly representative of the hardware we took special care in emulating the order of execution and data-flow as expected in hardware. An important part of this emulation was in the order of execution of the summation reduction tree present in the block-based matrix-vector computation.

The number of iterations required to attain a given accuracy is plotted in Fig. 8. Because the number of iterations required is data dependent [24], we have plotted the best, worst and median case. Note that in the worst case, there are systems that do not converge within the timeout given of 1000 iterations, when the mantissa width is below 13 bits.

5 Speed-up results

Overall performance comes from combining the (increasing) iteration time (Fig. 6) with the (decreasing) number of iterations (Fig. 8). Merging these data together we find the optimum mantissa-width that maximizes the speed of solution finding up-to the desired accuracy.

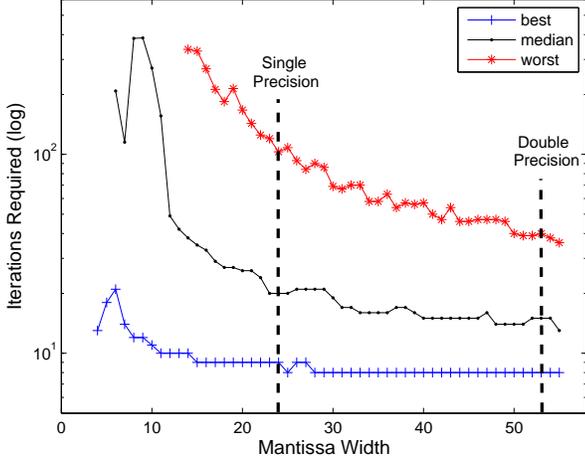


Fig. 8. MPC test-bench best, median, and worst iterations required to achieve an accuracy of 0.1 as function of mantissa width. The maximum iteration count was set at 1000.

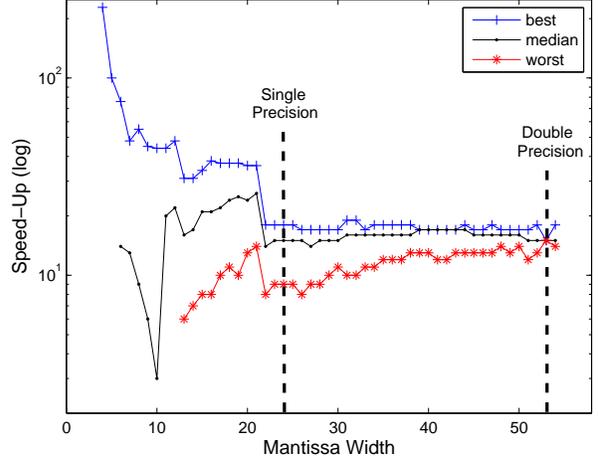


Fig. 9. Plot showing the worst, average, best and median speed-ups compared to a single high-end CPU running at 3.0 GHz using double precision.

We compare a high-capacity Xilinx Virtex5 330T FPGA with a high-performance CPU running at 3.0 GHz and using double precision floating point, with identical accuracy termination condition as the FPGA. These speed-up results are based on the following formulation (4), where I_{CPU} and I_{FPGA} correspond to the number of iterations, and T_{CPU} and T_{FPGA} correspond to the time required to complete an iteration on the CPU and FPGA, respectively.

$$\text{FPGA Speed-up} = \frac{\text{CPU}_{\text{time}}}{\text{FPGA}_{\text{time}}} = \frac{I_{\text{CPU}} \times T_{\text{CPU}}}{I_{\text{FPGA}} \times T_{\text{FPGA}}} \quad (4)$$

The combination of these data yields the main result: the ability to reduce mantissa compensates increased iteration count to produce faster solution computation. This speed-up, is shown in Fig. 9 for the best, average, worst and median cases compared to a high-end CPU. In this figure it is possible to observe the best overall speed-up of $228\times$ (for a 4-bit mantissa), the best overall speedup when even the worst case mantissa converged is $31\times$, an average speed-up of $14\times$ in single precision, the average speed-up of $26\times$ using a 21-bit mantissa, and speed-up of $15\times$ using double precision standard. Note that in all these cases the accuracy achieved is the same irrespective of mantissa width. Note also that all curves converge at double precision, as $I_{\text{CPU}} = I_{\text{FPGA}}$ in this case. In particular, we should note that both in the best case and in median case, a mantissa width of 21 performs better than both the double and single precision standard, with little to no penalty in the worst case.

6 Conclusions and Future Work

In this paper we have presented a study that trades-off computational precision by reducing mantissa width in order to produce a significant speed-up in iterative methods for solving systems of linear equations while keeping the same final solution accuracy. This is possible because there is a direct dependence between iteration count and computational precision in iterative methods for solving systems of linear equations.

This study is based on a test-bench from a Model Predictive Control problem, and empirically provides the optimal speed-up point where the reduced mantissa width maximizes overall performance. Beyond this point the increased mantissa width does not provide a significant iteration count reduction to justify the increased overheads, and before this point the improved performance does not outweigh the growth in required iteration count to reach the desired solution accuracy.

Using this fine-tuning scheme and applying a 21-bit mantissa, it is possible to achieve an average speed-up of $26\times$, a best case of $36\times$, a worst case of $14\times$, and a median of $26\times$ (compared to a high-end CPU running at 3.0 GHz) with all the problems reaching the same intended final solution accuracy

Future work will investigate data-dependent methods to achieve closer to median performance even in the worst case.

7 References.

- [1] N. Higham, *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA, USA, 2002.
- [2] Xilinx, “DS100 (v3.0) Virtex5 Family Overview,” <http://www.xilinx.com/>, Accessed on 1/03/2007.
- [3] P. Alfke, “CERN Scientists Use Virtex-4 FPGAs for Big Bang Research,” *Xcell Journal - Third Quarter*.
- [4] C. Leong, P. Bento, P. Rodrigues, J. Silva, A. Trindade, P. Lousa, J. Rego, J. Nobre, J. Varela, J. Teixeira and C. Teixeira, “Design and Test Issues of an FPGA Based Data Acquisition System for Medical Imaging Using PEM,” *IEEE Trans on Nuclear Science*, vol. 53, no. 3, pp. 761–769, June 2006.
- [5] R. Wei, M. Jin, J. Xia, Z. Xie and H. Liu, *Reconfigurable Parallel VLSI Co-Processor for Space Robots Using FPGA*. Proc. of IEEE Robotics and Biomimetics, 2006, pp. 374–379.
- [6] X. Tian and K. Benkrid, “Design and Implementation of a High-Performance Financial Monte-Carlo Simulation Engine of an FPGA Supercomputer,” in *Proc. of Field Programmable Technology*, 2008, pp. 81–88.
- [7] T. Güneysu, T. Kasper, M. Novotn, C. Paar and Andy Rupp, “Cryptanalysis with COPACOBANA,” *IEEE Trans on Computers*, vol. 57, no. 11, pp. 1498–1513, May 2008.
- [8] J. Maciejowski, “Predictive Control with Constraints,” Prentice Hall, Pearson Education Limited, Harlow, UK, 2001.
- [9] W. Zhang, V. Betz, and J. Rose, “Portable and Scalable FPGA-Based Acceleration of a Direct Linear System Solver,” in *Proc. of Field Programmable Technology*, 2008, pp. 17–24.
- [10] S. Haridas and S. Ziavras, “FPGA Implementation of a Cholesky Algorithm for a Shared-Memory Multiprocessor Architecture,” *Journal of Parallel Algorithms and Applications*, vol. 19, no. 6, pp. 411–226, Dec. 2004.
- [11] G. Morris and V. Prasanna, “An FPGA-Based Floating-Point Jacobi Iterative Solver,” in *Proc. of the 8th International Symposium on Parallel Architectures, Algorithms and Networks*, 2005, pp. 420–427.
- [12] A. Roldao, G. Constantinides and E. Kerrigan, “A Floating-Point Solver for Band Structured Linear Equations,” in *Proc. of Field Programmable Technology*, 2008, pp. 353–356.
- [13] D. Boland, and G. Constantinides, “An FPGA-based Implementation of the MINRES algorithm,” in *Proc. of Field Programmable Logic*, 2008, pp. 379–384.
- [14] R. Strzodka and D. Göttsche, “Pipelined mixed precision algorithms on FPGAs for fast and accurate PDE solvers from low precision components,” in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2006)*, Apr. 2006, pp. 259–268.
- [15] A. Buttari, J. Dongarra, J. Kurzak, P. Luszczek, and S. Tomov, “Using mixed precision for sparse matrix computations to enhance the performance while achieving 64-bit accuracy,” *ACM Trans. Math. Softw.*, vol. 34, no. 4, pp. 1–22, 2008.
- [16] J. Sun, G. Peterson, and O. Storaasli, “High-performance mixed-precision linear solver for fpgas,” *IEEE Trans. on Computers*, vol. 57, no. 12, pp. 1614–1623, 2008.
- [17] Y. Wang and S. Boyd, “Fast model predictive control using online optimization,” in *Proc. of Int. Federation of Automatic Control World Congress*, 2008, pp. 6974–6997.
- [18] T. Keviczky and G. J. Balas, “Receding horizon control of an F-16 aircraft: A comparative study,” *Control Engineering Practice*, vol. 14, no. 9, pp. 1023–1033, 2006.
- [19] H. Ferreau, P. Ortner, P. Langthaler, L. del Re, , and M. Diehl, “Predictive control of a real-world diesel engine using extended online active set strategy,” *Annual Reviews in Control*, vol. 31, no. 2, pp. 293–301, May 2007.
- [20] S. Boyd and L. Vandenberghe, *Convex Optimizations*. Cambridge University Press, United Kingdom, 2004, pp. 152–153.
- [21] A. Roldao and G. Constantinides, “A High Throughput FPGA-based Floating Point Conjugate Gradient Implementation,” in *Proc. of Applied Reconfigurable Computing*, 2008, pp. 75–86.
- [22] Xilinx, “Core Generator Floating Point v3,” http://www.xilinx.com/bvdocs/ipcenter/data_sheet/-floating_point_ds335.pdf, Accessed on 19/01/2009.
- [23] Collaborative Project, “Multi-precision Floating Point Library,” <http://www.mpfr.org/>, Accessed on 02/01/2009.
- [24] G. Meurant, *The Lanczos and Conjugate Gradient Algorithms from theory to Finite Precision Computation*. SIAM, Philadelphia, PA, USA, 2006, pp. 323–324.