2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines

Architectures and precision analysis for modelling atmospheric variables with chaotic behaviour

Francis P. Russell*, Peter D. Düben†, Xinyu Niu‡, Wayne Luk§, T. N. Palmer¶
Department of Computing, Imperial College London, London, SW7 2AZ, UK
{*francis.russell02**,*niu.xinyu10*‡,*w.luk*§}*@imperial.ac.uk*
Atmospheric Oceanic and Planetary Physics, University of Oxford, Oxford, OX1 3PU, UK
{*dueben*†,*t.n.palmer*¶}*@atm.ox.ac.uk*

***Abstract*—The computationally intensive nature of atmospheric modelling is an ideal target for hardware acceleration. Performance of hardware designs can be improved through the use of reduced precision arithmetic, but maintaining appropriate accuracy is essential. We explore reduced precision optimisation for simulating chaotic systems, targeting atmospheric modelling in which even minor changes in arithmetic behaviour can have a significant impact on system behaviour. Hence, standard techniques for comparing numerical accuracy are inappropriate. We use the Hellinger distance to compare statistical behaviour between reduced-precision CPU implementations to guide FPGA designs of a chaotic system, and analyse accuracy, performance and power efficiency of the resulting implementations. Our results show that with only a limited loss in accuracy corresponding to less than 10% uncertainly in input parameters, a single Xilinx Virtex 6 SXT475 FPGA can be 13 times faster and 23 times more power efficient than a 6-core Intel Xeon X5650 processor.**

I. Introduction

Climate and weather prediction are computationally intensive; even with high-performance computing resources, it is typically impossible to resolve important convective cloud systems in global models [1]. Numerical models of weather and climate show significant model error due to limited resolution and complexity, necessitating a need for even more resource-intensive models. Performance and power requirements for running such models with hard time constraints have led to the exploration of hardware accelerators to obtain greater throughput and power efficiency.

A common approach to enhance throughput of hardware designs is to reduce precision of computations so that additional hardware resources can be employed to increase parallelism. Excessive precision reduction however, reduces calculation quality and usefulness, so a trade-off must be made between performance and accuracy.

Lorenz showed that weather forecasting involves the prediction of a chaotic system [2]. This implies an exponential growth of errors in initial conditions; significant divergence between a CPU and hardware implementation is expected simply due to implementation differences, and is not in itself an indicator of error. Diagnostics that rely on solution convergence between implementations for validation are inappropriate here.

Short-term forecasts may have uncertainty in initial conditions due to the nature of measurement. Also, the response to a forcing (e.g. a change in CO_2 concentration) is difficult to predict and must be tested with numerical simulation. Given the chaotic nature of weather and initial condition uncertainty, one may consider a reduction in precision appropriate so long as the behaviour change introduced is acceptable compared to those introduced by other factors.

We investigate the reduction of precision in chaotic systems using the Lorenz 1996 model (a.k.a. Lorenz 1995), designed by Lorenz [3] to study interactions of atmospheric processes with non-linear, chaotic dynamics. Our analysis has applicability beyond this model – scale interactions within the Lorenz '96 model resemble scale interactive behaviour of various parts of numerical atmosphere models that is typically difficult to capture in idealised systems. Similar scale interactions are also important for turbulent energy cascades relevant to most applications in CFD (computational fluid dynamics). This paper contributes and presents:

- the hardware architecture of a two-scale Lorenz '96 simulation using Runge-Kutta time-stepping;
- a demonstration of how it is possible to make trade-offs between precision and throughput for a system where numerical divergence is expected;
- an analysis of the impact of varying the precision of variables at different scales in the Lorenz '96 implementation using error metrics appropriate to chaotic systems (the Hellinger distance);
- performance, precision and power consumption comparisons of reduced-precision FPGA implementations to an optimised CPU implementation.

II. Background

A. Lorenz '96

The two-scale Lorenz '96 model (hereafter called L96) was designed by Lorenz as a simple model to study predictability in a chaotic system. Despite its simplicity, L96 shares important properties of numerical atmosphere models such as scale interactions and chaotic behaviour. Also, its numerical

978-1-4799-9969-9/15 $31.00 © 2015 IEEE
DOI 10.1109/FCCM.2015.52

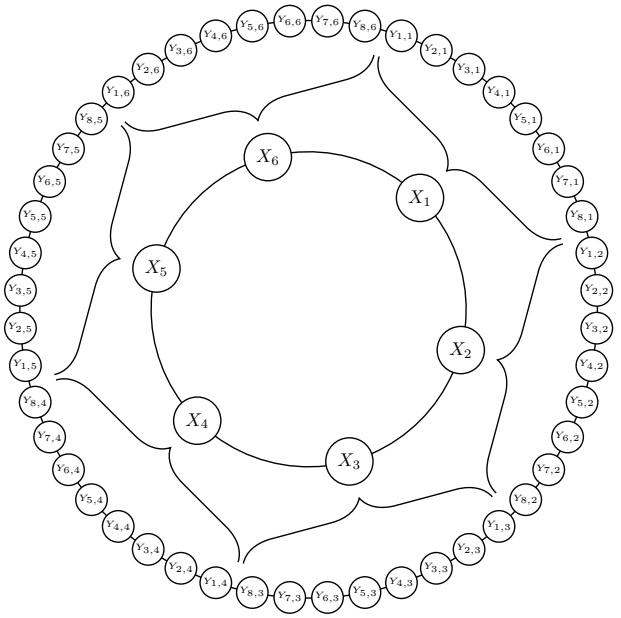


Figure 1. Visualisation of a two-scale L96 system with $J = 8$ and $K = 6$. Global-scale values (X_k) are updated based on neighbouring values and a reduction applied to the local-scale values ($Y_{j,k}$) associated with that value. Local-scale values are updated based on neighbouring values and the associated global-scale value. The neighbourhood topology of both the local and global-scale values is circular.

implementation has significant similarities to widely used CFD finite difference models in terms of the discrete grid point representation and time stepping schemes.

The model is defined by a set of coupled equations involving K global-scale variables ($X_{1\leq k\leq K}$), each of which is associated with a set of J local-scale variables ($Y_{1\leq j\leq J,k}$):

$$\frac{dX_k}{dt} = -X_{k-1}(X_{k-2} - X_{k+1}) - X_k + F - \frac{hc}{b}\sum_{j=1}^{J} Y_{j,k} \tag{1}$$

$$\frac{dY_{j,k}}{dt} = -cbY_{j+1,k}(Y_{j+2,k} - Y_{j-1,k}) - cY_{j,k} + \frac{hc}{b}X_k \tag{2}$$

where F is a forcing term, h is the coupling constant, b is the spatial-scale ratio and c is the time-scale ratio. Often $h = 1$, $F = 20$, $b = 10$ and $c = 4$ or $c = 10$.

Global and local-scale variables are arranged circularly (Fig. 1) such that $X_k = X_{k+K}$, $Y_{j,k} = Y_{j,k+K}$ and $Y_{j,k} = Y_{j-J,k+1}$. For clarity we show a small system; in practice we look at how to scale J and K to hundreds of thousands. We explore how to scale both J and K, but only run simulations with relatively small values of J (144) since local-scale values only influence global-scale values through a summation so larger values of J result in statistically similar behaviour.

For all simulations, we temporally discretise using Runge-Kutta. This has a significant impact on our accelerator design due to the number of intermediate terms required during each time-step.

B. Related Work

While FPGAs have been used in implementing chaotic systems, previous work has focused on security and digital communication [4]. This paper provides the first FPGA architecture and precision analysis for atmospheric modelling with chaotic behaviour.

Oriato et al. have accelerated the dynamical core of a Limited Area Model derived from the BOLAM model [5]. The model includes three-dimensional non-linear equations covering conservation of momentum, continuity and thermal energy. An explicit finite-difference scheme is used to solve these equations over a staggered C-grid. Parts of the computation are converted to use fixed point arithmetic.

Gan et al. have investigated a hybrid CPU-FPGA approach for computing the upwind stencil of the global shallow water equations [6] using a finite-volume and TVD Runge-Kutta discretisation on a cubed-sphere mesh. Range analysis is applied to kernels, and used to transform values to fixed point and reduced-precision floating point. Despite the stencil's small size, the high computational intensity leads to significant speedups in the FPGA implementation.

A major difference from our work is that the two studies above consider short-term simulations that show only limited propagation of model errors due to the chaotic dynamics of the atmosphere. We consider the influence of reduced precision on a long-term diagnostic for a system with strong chaotic behaviour.

Düben et al. have explored simulated stochastic processors and low-precision arithmetic in the context of atmospheric modelling for short [7] and long-term simulations [8]. Their two-level L96 model is run with bits in the mantissa randomly flipped after floating point operations, emulating a stochastic processor. Additionally, the dynamic core of the Intermediate Global Climate Model is run with both reduced precision, and with stochastic processor emulation for global simulations. Results suggest that imprecise computing strategies can be aggressively applied to small-scale dynamics without significantly altering those at larger scales.

III. Architecture

A. System Overview

The entire L96 Runge-Kutta update is moved to a hardware accelerator. The system state is copied from host memory to the accelerator's off-chip memory prior to computation. Arbitrary numbers of Runge-Kutta updates can be performed with the system state sent back to the host when needed.

All computations are moved to hardware, but host involvement remains during updates. Memory controller commands are sent by the host to the accelerator during computation since data access patterns for multiple iterations (Sec. III-G) are expensive to compute in hardware. The data-volume is small, so transfer time is not a limiting factor.

The structure of our design is shown in Fig. 2. The padding/stripping steps add/remove padding needed to satisfy

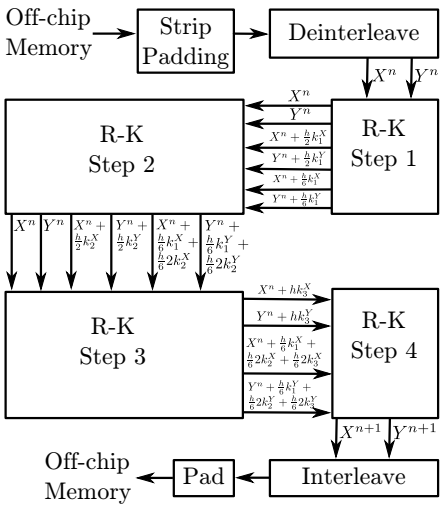


Figure 2. The hardware structure of a Runge-Kutta time-step for L96. X^n and Y^n represent the state of the system at time-step n. k_i^X and k_i^Y represent the i^{th} Runge-Kutta increment for the X and Y states, respectively. Expressions containing the factor $\frac{h}{6}$ represent intermediate values in the computation of $X^{n+1} = X^n + \frac{h}{6}(k_1^X + 2k_2^X + 2k_3^X + k_4^X)$ and $Y^{n+1} = \frac{h}{6}(k_1^Y + 2k_2^Y + 2k_3^Y + k_4^Y)$. All other expressions are Runge-Kutta estimated slopes.

memory controller constraints on read/write sizes. The deinterleave/interleave steps split/combine the X and Y state so that sending an X element can be synchronised with sending the first element of the associated Y state. Each Runge-Kutta step is implemented by a single kernel, computing the updated partial X and Y states and intermediate values.

B. In-memory Representation

We treat the state of a L96 system as a matrix (Fig. 3). The matrix is flattened to memory in column-major order so that all Y values associated with a particular X value are located contiguously in memory. We interleave the Y and X state such that each X-value is located before the associated Y values.

C. Data Path

For clarity, we describe the Runge-Kutta method for calculating y_{n+1} from y_n given the function f that calculates the gradient of y at t such that $\frac{dy}{dt} = f(t, y)$:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{3}$$

$$t_{n+1} = t_n + h \tag{4}$$

where:

$$k_1 = f(t_n, y_n) \tag{5}$$

$$k_2 = f(t_n + 0.5h, y_n + 0.5hk_1) \tag{6}$$

$$k_3 = f(t_n + 0.5h, y_n + 0.5hk_2) \tag{7}$$

$$k_4 = f(t_n + h, y_n + hk_3) \tag{8}$$

The kernels for each Runge-Kutta step are similar but differ in the number of inputs and outputs (e.g. all steps except the first take an input containing intermediate values

$$\begin{pmatrix} X_1 & X_2 & \cdots & X_K \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ Y_{1,1} & Y_{1,2} & \cdots & Y_{1,K} \\ Y_{2,1} & Y_{2,2} & \cdots & Y_{2,K} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{J,1} & Y_{J,2} & \cdots & Y_{J,K} \end{pmatrix}$$

Figure 3. State of the L96 system arranged as a matrix. Each column contains a single X elements and all Y elements associated with the X element. Each column may contain zero or more padding elements after the X element due to vectorisation (Sec. III-F).

of X^{n+1} and Y^{n+1}). Each step is specialised to avoid unnecessary computation and communication. X and Y remain deinterleaved during processing, but are synchronised so that each X entry is sent/received by kernels simultaneously with the first element of the associated Y-values.

D. Precision

We analyse the effect of precision reduction at the level of global and local-scale values of L96. In weather and climate modelling, it is important to represent large-scale patterns (analogous to global-scale quantities in L96) accurately, but of lesser importance to represent small-scale dynamics (analogous to local-scale quantities) in high accuracy. Small-scale dynamics are inherently uncertain due to the strong influence of parametrisation schemes and viscosity in atmosphere models and initial value uncertainty. Therefore, precision of local-scale quantities should be chosen to be equal to or less than the precision of global-scale quantities [8].

In our design, all values are stored in off-chip memory as IEEE 754 single-precision floating-point values. Global and local-scale quantities are converted to and from lower precision in the "deinterleave" and "interleave" kernels, respectively (Fig. 2).

All intermediate values such as derivatives and slope estimates are maintained in the same precision as the associated X or Y state. For the summation of Y elements needed by the X-derivative, we use an accumulator of the same type as the X elements for improved precision.

E. Boundary Conditions

Calculation of X and Y derivatives both involve stencil operations with circular boundary conditions. Both stencils are small, however:

- Data dependencies force output to be delayed and reordered with respect to the stencil input. Since we pass the original input and Runge-Kutta intermediate values through the pipeline, these must be reordered and delayed in an identical manner.
- Since we couple X and Y, any buffering of X elements requires buffering of entire columns of Y. We wish to avoid this as much as possible since the required buffer size will increase with K.

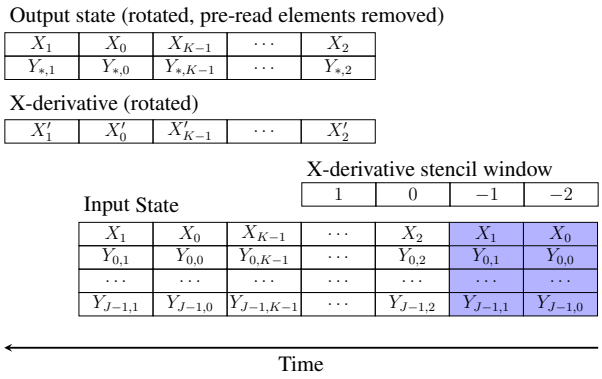


Figure 4. Time at which columns of the input state are processed and output produced for a stencil operation over X. The pre-read region (shaded) is used to calculate the X-derivative stencil with the Y elements discarded. The original state is output, but rotated in the same manner as the X-derivative with pre-read elements removed. Look behind causes the final output elements to be delayed by one column relative to the final input element.

We consider calculation of a stencil requiring S_n elements applied to a stream of n elements where $n \gg S_n$ and we wish to store as few elements as possible. We call the S_f elements ahead of those being processed "look-ahead" and the S_b after as "look-behind" such that $S_n = S_b + 1 + S_f$.

1) Look-ahead: Look-ahead makes it impossible to output updated elements in the same order as they are received. We start producing elements as soon as we receive the first S_n and output X and Y elements in a "rotated order" with respect to the input.

Calculation of the final elements of the output requires elements from the start of the stream. For Y updates, we can store the initial S_f elements from the stream and use them later. For X updates, storing initial elements requires storing entire columns of Y. To avoid this, we implement a "pre-read" approach whereby initial columns of the input-state are read both at the start and end of an update. The first pass over the 'pre-read" region uses the X values but discards Y; the second pass uses both X and Y.

For the stencil over X, $S_f = 2$. Since each Runge-Kutta intermediate step introduces a rotation, we need to pre-read 8 columns of the input state. The rotation to X is undone by the host when the state is copied back.

2) Look-behind: A stencil with only look-behind does not introduce rotations, but the output is still delayed until sufficient elements are received to evaluate the stencil. Since we need to maintain synchronisation between input and output, S_b columns of the input must be buffered.

We illustrate the effect of look-ahead and look-behind on the X-derivative stencil in Fig. 4. Buffering of state due to look-ahead is eliminated via pre-reading but buffering due to look-behind cannot be avoided.

F. Vectorisation

To maximise throughput we vectorise handling of the Y state (by creating multiple in-kernel data-paths). Alternatively we could have replicated the Runge-Kutta pipeline, but the vectorisation approach has a number of benefits.

Our design is constructed such that we can choose an arbitrary vector length v_n representing the number of elements read from a Y-related stream in a single cycle. We enforce the requirement that J is a multiple of v_n.

The arithmetic required to implement the Y-derivative calculation is duplicated v_n times. Since we do not duplicate the pipeline, we generate no additional counters nor communication channels between kernels. Most importantly, the sizes of the buffers required to store and delay Y-related values are nearly independent of v_n so increasing v_n has minimal effect on on-chip memory usage.

Calculation of the X-derivative involves a summation over each column of Y. We implement an accumulator that accepts one element per cycle, which is relatively costly in terms of adders. With vectorisation we avoid duplicating the accumulator; instead we use another $v_n - 1$ adders to sum the vector elements for input to the accumulator.

One overhead incurred is that each X-state element must be padded to v_n elements. For an arbitrary system, this means that the fraction of padding of an in-memory representation of the system will be $\frac{v_n - 1}{J+1}$. For a typical value of $J = 128$, a system where $v_n = 8$ will consist of 5.4% padding and where $v_n = 16$, it will have 11.6% padding.

G. Multiple Iterations

We can perform multiple L96 time-steps with only a single invocation from the host machine. As described in Section III-E, a "pre-read" region is read at both the start and end of a time-step update. Therefore, the pre-read region cannot be overwritten until it has been read both times.

The memory controller requires all reads and writes to have sizes and occur at addresses which are multiples of a *burst size* (384 bytes). During a time-step update, the new state is written starting at the first burst that does not contain any pre-read data, causing the start of the L96 state to move forward in memory each time-step.

The L96 state is padded to the memory controller's burst size and circular addressing is used such that the flattened representation of the system state (including padding) "rotates" in memory by multiples of the burst size; rotation is undone by the controller on state transfer to the host. Fig. 5 illustrates the memory layout transformation caused by the first time-step.

IV. Precision Analysis

A. Metrics

L96 exhibits complex wave-like and chaotic behaviour [9]. Specifically, small perturbations in the initial conditions of the system will result in significant differences in its evolution. Consequently, we cannot use a direct comparison of state between hardware and CPU implementations after a large number of time steps due to the expected divergence.

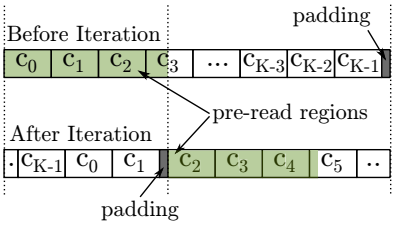


Figure 5. Rotation of the L96 system state caused by a single time-step update. We show a rotation by 3 columns rather than the actual 8. Dashed lines indicate burst size aligned addresses. The updated state is written in front of the pre-read region causing the flattened state (including padding) to rotate in-place. The pre-read region size is chosen to be the minimum burst size multiple capable of holding the number of columns to be pre-read.

The climatology of L96 is typically defined as the probability density function (PDF) of the X variables, averaged over a long run: 10,000 model time units, equivalent to more than 100 "atmospheric years" [3]. For assessing simulation accuracy, we compare the PDFs of both local and global-scale variables.

A metric often used to compare PDFs is the *Hellinger distance* (see for example [10]). For two probability distributions p and q, the Hellinger distance H is defined as:

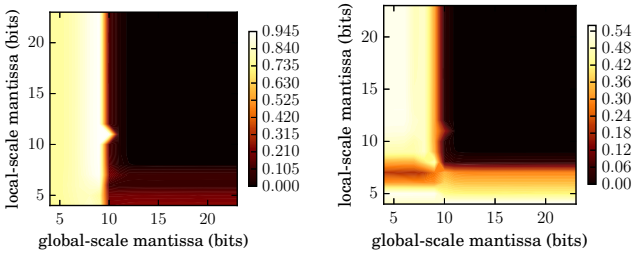
$$H(p,q) = \sqrt{\frac{1}{2}\int \left(\sqrt{p(x)} - \sqrt{q(x)}\right)^2 dx} \tag{9}$$

Taking the Hellinger distance between a high-precision CPU implementation and reduced-precision implementations enables us to determine the quality of the simulation. As a baseline, we calculate the Hellinger distance between two high-precision CPU implementations with different starting states. Since the Hellinger distance is a measure of long-term dynamics, a change in initial conditions should not change the PDF and the Hellinger distance should be close to zero. The Hellinger distance between simulations that only differ in the initial conditions can serve as an estimate of the uncertainty in the measurement of the Hellinger distance for a given model setup.

B. Precision Reduction

Reducing calculation precision enables the reduction of hardware resource utilisation, permitting instantiation of additional functional units for increasing throughput. For the L96 simulations, we wish to determine the extent to which we can increase throughput while maintaining an acceptable level of accuracy.

We only consider reduced precision floating point and do not attempt to use fixed point. We do not attempt to optimise each intermediate value in our pipeline; rather, we investigate the effects of precision reduction on terms associated with the global and local-scale quantities of L96. Düben et al. have shown for an atmosphere model that the precision to calculate small-scale dynamics can be reduced much further than the precision for large-scale dynamics with a smaller influence on model results and forecast quality [8].



(a) global-scale distance (b) local-scale distance

Figure 6. Hellinger distances (smaller is better) for X and Y state values between a double-precision CPU implementation and the CPU implementation using truncated precision for local and global-related values. $F = 40$, $b = 10$, $c = 10$, $J = 64$, $K = 2000$ and the simulation was run for 3,750,000 time-steps with the state sampled every 1000 time-steps.

C. Implementation of Analysis

We extended the L96 C++ implementation to support reduced precision. To match the hardware implementation, we require behaviour equivalent to IEEE 754 with round-to-nearest, no denormalised value support and non-standard mantissa lengths. We originally used an arbitrary precision library but required higher-performance for long runs so we approximated reduced precision with round-to-nearest, tie-to-even behaviour through bit manipulation of double-precision values. Since we have no access to the "sticky" bits of the underlying implementation, we will sometimes invoke tie behaviour when a correct implementation would round. We note that the rounding mode has significant impact on the behaviour of this model at low precisions; using either truncation or round-to-nearest, tie away from zero rounding behaviour altered results significantly.

Since the hardware and reduced-precision CPU implementations differ in order and implementation of arithmetic operations, the results from each will diverge. Also, the chaotic nature of L96 causes even tiny perturbations to lead to divergence. However, we can use the CPU implementation as a tool to analyse the effect of different local and global-scale value mantissa widths on the Hellinger distance.

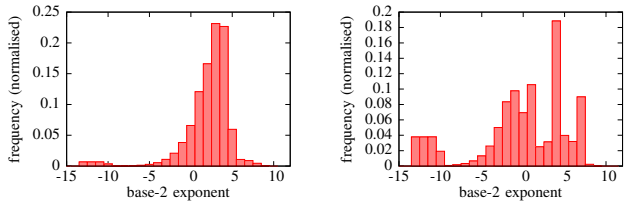
We plot the Hellinger distances between the PDFs of the elements of the global and local-scale states against a double-precision implementation (Fig. 6). We see expected behaviour – that reduced precision leads to greater error. Although the probability distribution of global-scale values appears minimally affected by the use of small (≤ 8 bits) mantissas for local-scale values, this is misleading since the probability distribution of local-scale values has been significantly altered. The results also confirm previous work [8] that concludes the precision of values at different scales should be optimised independently, which is important for scaling our analysis to more complex models.

To determine the extent to which precision can be reduced, we must choose an acceptable Hellinger distance. We calculate Hellinger distances between the double-precision reference, the same with changed initial conditions, and with

Table I
HELLINGER DISTANCES (4 S.F.) BETWEEN A DOUBLE-PRECISION SIMULATION AND THE SAME WITH DIFFERENT INITIAL CONDITIONS (AVERAGED FROM FIVE RUNS) AND ALTERED PARAMETERS.

Run	H (global)	H (local)	Est. min. mantissa (global)	Est. min. mantissa (local)
Changed initial conditions	2.788e-3	1.939e-4	15	12
$c \times 0.99$, $F \times 0.99$	4.605e-3	1.505e-3	13	10
$c \times 1.01$, $F \times 1.01$	5.042e-3	1.719e-3	13	10
$c \times 0.9$, $F \times 0.9$	4.047e-2	1.625e-2	11	9
$c \times 1.1$, $F \times 1.1$	3.620e-2	1.415e-2	11	9

We estimate minimum mantissa widths required to produce a simulation with smaller global and local-scale Hellinger distances based on the results from CPU simulation. Where multiple combinations of global and local-scale mantissa lengths satisfy the requirements, we choose the configuration with the smallest combined mantissa length.



(a) global-scale (b) local-scale

Figure 7. Exponent (base-2) distributions for global and local-scale values derived from CPU simulation with $F = 40$, $b = 10$ and $c = 10$. The profile includes all intermediate values calculated in the specified precision.

c and F altered by 1% and 10% (Table I). Model parameters always have a degree of uncertainty, and we consider 1% to be realistic for most numerical models in Computational Fluid Dynamics and 10% to be reasonable within atmospheric modelling [11].

Runtime profiling of floating-point exponent values in the CPU simulation was performed for 8000 time-steps (Fig. 7). Results suggest that exponent values of up to 10 and 11 are required to capture the largest global and local-scale values, respectively. Exponent widths of 5-bits for both global and local-scale values can represent the majority ($\geq 99.999\%$) of values generated during a simulation.

V. RESULTS

Hardware simulations were performed on a Maxeler MAX3A Vectis Dataflow Engine (DFE) which contains a Xilinx Virtex 6 SXT475 FPGA and 24GB of DD3 RAM. Connection to the host was via PCI express. All designs were compiled to run at 150MHz.

To compare performance between DFE and CPU, we wrote an optimised L96 simulation in C++. OpenMP was used for intra-node parallelisation and NUMA functionality employed to prevent thread migration between physical cores. The code was compiled with the Intel C++ Compiler 12.1.4. CPU simulations were run on the DFE host machine which contained two 6-core Intel Xeon X5660s running at 2.67GHz, each having 12MB L3 cache (System 2 in Sec. V-C). Hyper-threading was found to be beneficial so two threads were assigned to each physical core. The host was running Centos Linux 6.3.

Table II
PRECISION, VECTOR WIDTHS AND RESOURCE UTILISATION OF L96 DFE BUILDS.

Build	Global-scale type	Local-scale type	Vector width	Utilisation (%)		
				Logic	DSP	BRAM
DFE1	float(8, 24)	float(8, 24)	8	55.00	48.16	24.25
DFE2	float(6, 15)	float(5, 12)	16	69.85	32.84	28.67
DFE3	float(5, 13)	float(5, 10)	16	64.28	32.84	27.63
DFE4	float(5, 13)	float(5, 10)	24	79.15	47.92	33.74

Table III
HELLINGER DISTANCES (4 S.F.) AGAINST A DOUBLE-PRECISION CPU IMPLEMENTATION.

Build	$J = 64$		$J = 144$	
	H (global)	H (local)	H (global)	H (local)
Changed initial cond.	2.788e-3	1.939e-4	7.029e-3	6.291e-4
$c \times 0.99$, $F \times 0.99$	4.605e-3	1.505e-3	1.399e-2	1.726e-3
$c \times 1.01$, $F \times 1.01$	5.042e-3	1.719e-3	1.067e-2	1.861e-3
$c \times 0.9$, $F \times 0.9$	4.047e-2	1.625e-2	1.167e-1	1.487e-2
$c \times 1.1$, $F \times 1.1$	3.620e-2	1.415e-2	8.826e-2	1.236e-2
DFE1	2.934e-3	2.881e-4	7.472e-3	1.180e-3
DFE2	2.776e-3	2.707e-4	4.320e-2	2.443e-3
DFE3	3.267e-3	6.742e-4	7.289e-2	1.012e-2
DFE4	N/A	N/A	7.404e-2	1.005e-2

$F = 40$, $b = 10$, $c = 10$, $J = 120$, $K = 2000$ and the simulation was run for 3,750,000 time-steps with the state sampled every 1000 time-steps. Results are provided for both $J = 64$ (as in Table I) and $J = 144$. Since J must be a multiple of the vector width we cannot calculate Hellinger distances for DFE4 where $J = 64$.

Four DFE designs were built for benchmarking and precision analysis: one at single precision, one at the precision estimated necessary for similar accuracy to single-precision and two at the precision level estimated to have errors of the same order of magnitude as those caused by 1% uncertainly in the input parameters. To enable comparison with the Hellinger distances in Table I, design DFE3 was built with a reduced vector width. DFE3 can be run with $J = 64$ permitting comparison with the CPU simulations but DFE4 cannot since J is not a multiple of the vector width.

All designs (except DFE3) were compiled with the maximum supported vector width (Table II). The notation float(e, m) denotes a floating point type with an e-bit exponent and m-bit mantissa, including the implicit bit. The maximum value of J each design could support was reduced to 512 since larger values are of minimal interest to climate scientists and this improves the chances of routing. DFE2 was compiled with a 6-bit exponent for global-scale values due to restrictions on mantissa-exponent size combinations imposed by the Maxeler tools. We compiled a design using float(5,13) for both global and local-scale variables with a vector width of 16 and a maximum value of J of 128000, demonstrating we can scale J much higher if necessary.

A. Precision

We calculate the Hellinger distances for the local and global-scale state variables between each DFE implementation and a double-precision CPU implementation (Table III).

For $J = 64$, DFE1 and DFE2 have Hellinger distances similar to those seen between double-precision simulations

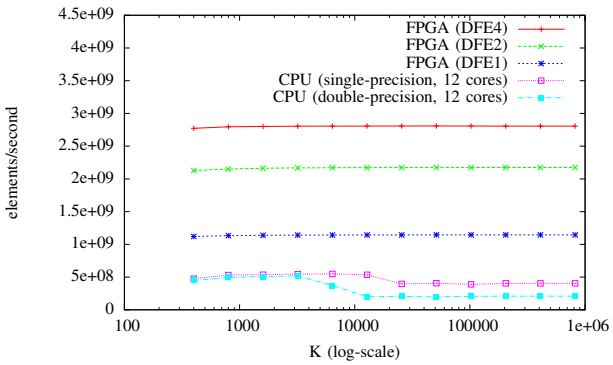


Figure 8. Throughput of CPU and DFE implementations in elements/second for $J = 144$ and varying values of K. CPU simulations were run in both single and double precision. The DFE simulations were compiled using the configurations in Table II.

Table IV
SPEED-UP FACTORS.

Build	Speedup (3 s.f.)	
	CPU (single, 1 core)	CPU (single, 12 cores)
CPU (single, 1 core)	1.00	0.0954
CPU (single, 6 cores)	5.30	0.506
CPU (single, 12 cores)	10.5	1.00
CPU (double, 1 core)	0.921	0.0879
CPU (double, 6 cores)	2.70	0.258
CPU (double, 12 cores)	5.39	0.514
DFE1	29.6	2.83
DFE2	56.3	5.37
DFE4	72.7	6.93

Factors are relative to a single-precision CPU implementation running with a single core and 12 cores (two threads per hyper-threaded physical core). $J = 144$ and $K = 819200$.

with changes to the initial conditions. DFE3, which has the lowest precision exhibits a Hellinger distance comparable to that caused by a 1% variation in c and F (Table I). We also show Hellinger distances for $J = 144$ (the least common multiple of vector widths used that is ≥ 128). For DFE4 (with $J = 144$), our Hellinger distances are outside those expected by varying input parameters by 1% (we expect larger distances due to having optimised for a different J) but within those expected by 10% variation.

B. Performance

We compare performance between the C++ implementation running on different numbers of cores in single and double-precision and different DFE builds (Fig. 8) with speedup factors shown in Table IV. DFE3 is excluded since it has the same precision as DFE4, but lower throughput due to the smaller vector width.

Our results show that a single DFE running with reduced-precision can outperform a 6-core Xeon X5660 running a single-precision implementation by over 13 times, and over 6 times when using two sockets.

For problems where $K \leq 6400$, the single and double-precision CPU implementations have near-identical performance. These computations appear arithmetic-bound and manual vectorisation would likely increase performance for these sizes. At $K = 6400, J = 144$ the size of a double precision L96 system state is just over 7MB which can reside within the 12MB L3 cache of a single socket. Larger systems no longer fit into cache and the single-precision implementation achieves ≈ 2 times the performance of the double-precision one for these sizes, suggesting they are bandwidth limited.

Our CPU implementation scales well from 6 to 12 cores. To achieve this, we use a "first-touch" policy for initialising the L96 state so memory is allocated in the NUMA domain of each thread. Prior to implementing this, performance improvement when moving from 6 to 12 cores was minimal, also indicating the implementation is bandwidth limited for large sizes.

Table V
SYSTEM THROUGHPUT, TOTAL POWER CONSUMPTION AND EFFICIENCY (3 S.F.) FOR CPU AND DFE IMPLEMENTATIONS

Build	Throughput (elements/s)	Power (W)	Efficiency (elements/J)	Relative Efficiency
System 1 (4-cores)	1.63e8	208 (228)	7.83e5	0.972
System 2 (6-cores)	2.05e8	399 (477)	5.14e5	0.638
System 2 (12-cores)	4.05e8	503 (581)	8.05e5	1.00
DFE1 (System 1)	1.14e9	137	8.35e6	10.4
DFE2 (System 1)	2.17e9	143	1.52e7	18.9
DFE4 (System 1)	2.81e9	146	1.92e7	23.9

Throughput is specified in updated elements per second where an element is a single scalar from the L96 state produced by a full Runge-Kutta update. We subtract estimated static power requirements of unused Maxeler cards from our power values (original values in parentheses). CPU implementations were run with two threads per hyper-threaded physical core. Relative efficiency uses the most power efficient CPU execution as the baseline.

C. Power Consumption

We measure the power requirements of the CPU and FPGA implementations on two systems:

1) A hyper-threaded four-core Intel Core i7 870 running at 2.93 GHz with 8MB L3 cache, 16GB RAM and a single MAX3A Vectis card. Total idle power consumption of 93W.
2) A dual-socket machine with each socket containing a hyper-threaded six-core Intel Xeon X5650 running at 2.67 GHz with 12MB L3 cache and four MAX3A Vectis cards. The system had 48GB of RAM with 24GB in each socket's NUMA domain. Total idle power consumption of 373W.

Total workstation power consumption was measured when running each implementation in different configurations (Table V). We subtract the static power requirements of the unused Maxeler cards from our reported figures (the average reported static power requirement per card is 19.6W).

In terms of computation per Watt, the reduced precision designs achieve power efficiency of up to 23 times higher than the most efficient CPU-only system. Assuming perfect scaling (not unreasonable considering the need in weather forecasting to calculate *emsembles*) system 1 could reach

Table VI
READS AND WRITES PERFORMED ON INPUT, OUTPUT AND INTERMEDIATE STATE BUFFERS BY EACH RUNGE-KUTTA STEP IN THE CPU IMPLEMENTATION.

Runge-Kutta Step	State Value					
	Input		Output		Intermediate	
	Read	Written	Read	Written	Read	Written
1	✓			✓		✓
2	✓		✓	✓	✓	✓
3	✓		✓	✓	✓	✓
4			✓	✓	✓	

a relative efficiency of 41 times if it contained three fully utilised MAX3A cards.

D. Discussion

Excluding data transfer due to pre-reading and padding, the amount of data read from and written to off-chip memory for a complete time-step is close to the L96 state size (a theoretical minimum). In contrast, the CPU implementation reads 9 times and writes 7 times the state size, respectively. The CPU implementation has a working set of approximately 3 times the state size consisting of storage for the input and output states, and a Runge-Kutta intermediate value. The buffers each Runge-Kutta step accesses are shown in Table VI.

Each buffer access in Table VI can be mapped to the corresponding inter-stage links in the hardware implementation (Fig. 2). However, the pipelined nature of the implementation means that the state passed between Runge-Kutta steps is never stored completely. Parallelisation occurs across the entire Runge-Kutta update as kernels operate on all four steps simultaneously.

In a CPU implementation, the natural approach involves domain decomposition to reduce inter-core communication. Parallelisation occurs across each Runge-Kutta step, with each thread updating partitions of the state. However, all threads work on the same Runge-Kutta step simultaneously. Consequently, communication between each Runge-Kutta step must occur via main memory (for large systems).

We consider the reduced memory bandwidth required for our DFE implementations to be a natural consequence of being able to express the Runge-Kutta update as a pipeline. Reducing the bandwidth requirements of the CPU implementation would most likely require some sort of iteration space partitioning beyond the scope of current compilers, or cause a significant increase in code complexity if implemented by hand.

VI. CONCLUSION

We designed a hardware architecture for the chaotic two-scale L96 system. We built FPGA implementations at different precision levels, guided by our CPU-based analysis of Hellinger distances and showed that acceptable behaviour can be achieved at precision significantly lower than single-precision. We demonstrated how to reduce precision of a chaotic system to improve throughput of a hardware implementation.

The performance improvements achieved have enabled climate scientists to substantially increase the scales of analyses involving the L96 system with and without reduced-precision. Previous work in this area was limited to using simulations to explore the latter [8].

Current and future work includes extending our research to cover more realistic models, supporting systems with multiple accelerators, and exploring techniques and tools to automate the architecture optimisation and precision analysis steps in our approach.

VII. ACKNOWLEDGEMENT

This work is supported in part by the European Union Seventh Framework Programme under grant agreement numbers 257906, 287804, 291406 and 318521, by the UK EPSRC, by the Maxeler University Programme, by the HiPEAC NoE, by Altera, and by Xilinx.

REFERENCES

[1] J. Shukla *et al.*, "Toward a New Generation of World Climate Research and Computing Facilities," *Bulletin of the American Meteorological Society*, vol. 91, no. 10, pp. 1407–1412, 2010.

[2] E. N. Lorenz, "Deterministic nonperiodic flow," *Journal of the Atmospheric Sciences*, vol. 20, no. 2, pp. 130—-141, 1963.

[3] ——, "Predictability – a problem partly solved," in *Predictability of Weather and Climate*, T. Palmer and R. Hagedorn, Eds. Cambridge University Press, 2006, pp. 40–58.

[4] L. Merah *et al.*, "Design and FPGA implementation of Lorenz chaotic system for information security issues," *Applied Mathematical Sciences*, vol. 7, no. 5, pp. 237–246, 2013.

[5] D. Oriato *et al.*, "Acceleration of a meteorological limited area model with dataflow engines," in *Application Accelerators in High Performance Computing (SAAHPC), 2012 Symposium on*, Jul. 2012, pp. 129–132.

[6] L. Gan *et al.*, "Accelerating solvers for global atmospheric equations through mixed-precision data flow engine," in *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, Sep. 2013, pp. 1–6.

[7] P. D. Düben and T. N. Palmer, "Benchmark tests for numerical weather forecasts on inexact hardware," *Mon. Wea. Rev.*, vol. 142, pp. 3809–3829, 2014.

[8] P. D. Düben *et al.*, "The use of imprecise processing to improve accuracy in weather & climate prediction," *Journal of Computational Physics*, vol. 271, pp. 2–18, 2014.

[9] S. Herrera *et al.*, "Spatio-temporal error growth in the multi-scale Lorenz'96 model," *Nonlinear Processes in Geophysics*, vol. 17, no. 4, pp. 329–337, 2010.

[10] H. M. Arnold *et al.*, "Stochastic parametrizations and model uncertainty in the Lorenz'96 system," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 371, no. 1991, p. 20110479, 2013.

[11] L. Tomassini *et al.*, "On the connection between tropical circulation, convective mixing, and climate sensitivity," *Quarterly Journal of the Royal Meteorological Society*, 2014.