Architectures and precision analysis for modelling atmospheric variables with chaotic behaviour

FCCM 2015

Francis P. Russell[†]

Peter D. Düben[‡] Xinyu Niu[†] Wayne Luk[†] Tim N. Palmer[‡]

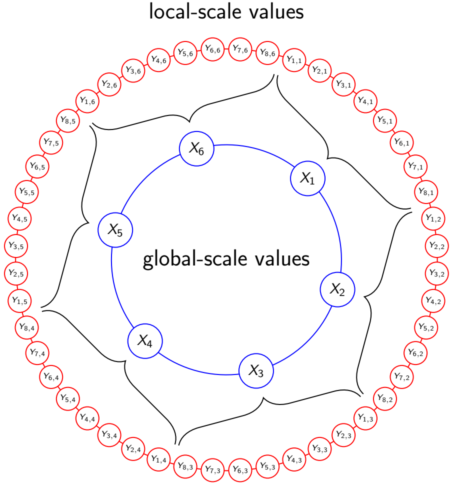
Imperial College London[†] & Oxford University[‡]

05/05/2015

Weather Modelling

- Atmospheric modelling requires simulation of a chaotic system.
- Chaotic systems are highly sensitive to initial conditions - *the butterfly effect*.
- There are many sources of uncertainly in atmospheric simulations.
- Sets of multiple simulations, called *emsembles* are run to build a distribution of outcomes.
- *Lorenz'96* is a simple model designed by Lorenz to study predictability in chaotic systems.
- We study the *two-scale* Lorenz'96 system which models small and large-scale dynamics.

The Lorenz'96 system (K=6, J=8)



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

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

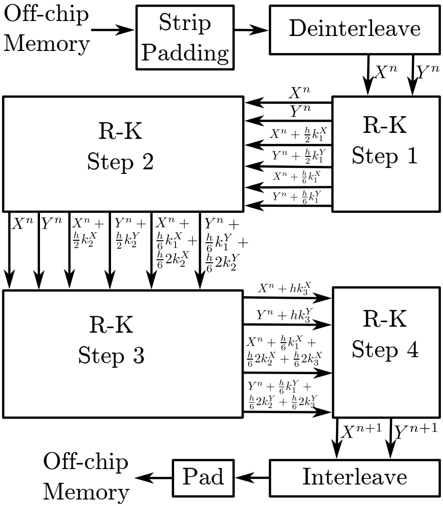
Typically, the largest J of interest ≈ 128. We increase the value of K when increasing system size (up to $\approx 800{,}000$).

Contributions

- A hardware design of the two-scale Lorenz'96 system, useful for studying the effects of multi-scale interactions in weather modelling.
- Demonstration of how trade-offs can be made between precision and throughput for the hardware implementation of a system with chaotic behaviour.
- Present an analysis of the precision reduction impact on a hardware implementation of Lorenz'96 using metrics appropriate for chaotic systems.
- Show the effects on power and performance, and compare to an optimised CPU implementation.

Hardware Design

- Pipelined Runge-Kutta timestepping.
- Different precisions for global and local-scale quantities.
- Local-scale values processed with customisable vector width.
- Periodic boundary conditions are problematic for a streaming design.
- Periodic properties of the system are used to enable in-place updates, “rotating” the system each update.



How much precision is enough?

- Computational scientists typically only have to choose between single and double precision.
- Choosing double precision is the simplest way to avoid thinking about the way individual degrees of freedom have been discretised.
- We can usually compare a reduction precision implementation against an analytical solution for simple test cases and a high-precision implementation for more complex ones.

Neither of these approaches is directly applicable to chaotic systems due to the Butterfly effect.

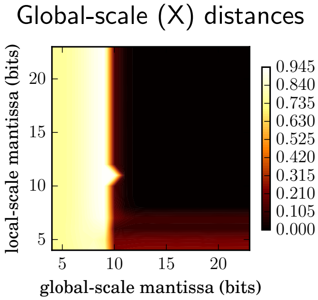
Error metrics

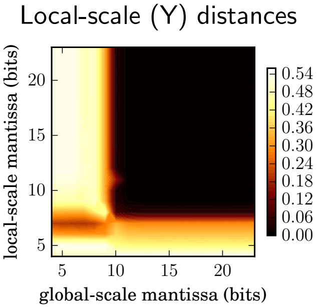
- We cannot compare long runs of Lorenz'96 at different precisions using standard error metrics.
- Even when starting from the same initial conditions, we expect the chaotic nature to magnify effects of different number representations, order of operations and other implementation artefacts.
- We use the *Hellinger distance* to compare the probability density functions of the global and local-scale values of the simulation.
- For two probability density functions p and q, the Hellinger distance is defined as:

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

Estimating the error

Using an arbitrary precision CPU-implementation we vary the precision of local and global-scale associated values. Distances are compared against those of a double-precision implementation.





Precision changes in variables at one scale do not typically affect those at the other, so we could optimise them *independently*.

Estimating appropriate mantissa sizes

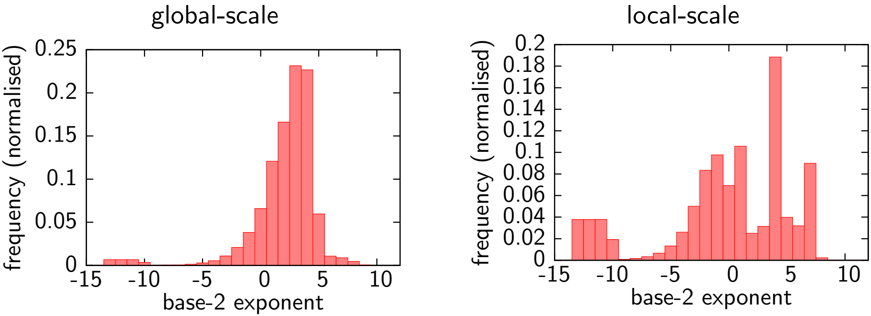
Using the CPU-based reduced-precision analysis, we can calculate minimum mantissa widths for local and global-scale values, depending on the amount of error we are willing to accept.

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

Scaling c and F represents variation in the input parameters by 1 and 10% due to measurement uncertainty.

Profiling exponent ranges

We also use the CPU-based implementation to profile exponent ranges for the global and local-scale values.



Most exponents are ≤ 9 so we need at least 5 bits for both global and local-scale (signed) exponent values.

The resulting designs

We build designs[1] at different precisions, allowing us to process different number of input elements per cycle (the vector width).

- **DFE1** Uses single precision.
- **DFE2** Estimated precision for minimal effect on result.
- **DFE3[2]/4** Estimated as similar error to 1% variation in input parameters.

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

[1] Maxeler MAX3A Vectis Dataflow Engine (Xilinx Virtex6 SXT475).

[2] Used to facilitate accuracy comparisons with the CPU-based implementation.

Precision Results

For $J = 64$ (which the CPU-simulations were run with) we have similar errors to those predicted by the CPU-simulations. For $J = 144$, we have slightly larger errors, but still acceptable for use.

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 (g=f(8,24), l=f(8,24), w=8)	2.934e-3	2.881e-4	7.472e-3	1.180e-3
DFE2 (g=f(6,15), l=f(5,12), w=16)	2.776e-3	2.707e-4	4.320e-2	2.443e-3
DFE3 (g=f(5,13), l=f(5,10), w=16)	3.267e-3	6.742e-4	7.289e-2	1.012e-2
DFE4 (g=f(5,13), l=f(5,10), w=24)	N/A	N/A	7.404e-2	1.005e-2

Precision Results

For $J = 64$ (which the CPU-simulations were run with) we have similar errors to those predicted by the CPU-simulations. For $J = 144$, we have slightly larger errors, but still acceptable for use.

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 (g=f(8,24), l=f(8,24), w=8)	2.934e-3	2.881e-4	7.472e-3	1.180e-3
DFE2 (g=f(6,15), l=f(5,12), w=16)	2.776e-3	2.707e-4	4.320e-2	2.443e-3
DFE3 (g=f(5,13), l=f(5,10), w=16)	3.267e-3	6.742e-4	7.289e-2	1.012e-2
DFE4 (g=f(5,13), l=f(5,10), w=24)	N/A	N/A	7.404e-2	1.005e-2

Precision Results

For $J = 64$ (which the CPU-simulations were run with) we have similar errors to those predicted by the CPU-simulations. For $J = 144$, we have slightly larger errors, but still acceptable for use.

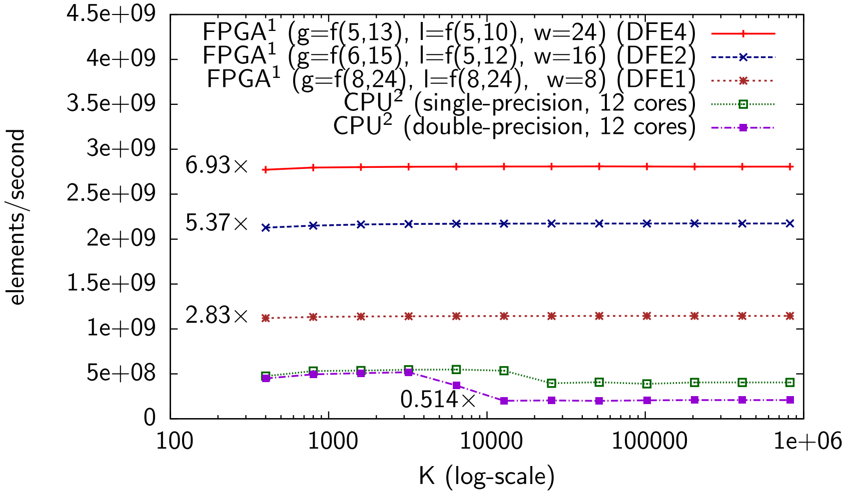
Build	$J = 64$		$J = 144$	
	H (global)	H (local)	H (global)	H (local)
Changed initial cond.	2.788e-3	1.939e-4	7.029e-3	6.291e-4
$c \times 0.99$, $F \times 0.99$	4.605e-3	1.505e-3	1.399e-2	1.726e-3
$c \times 1.01$, $F \times 1.01$	5.042e-3	1.719e-3	1.067e-2	1.861e-3
$c \times 0.9$, $F \times 0.9$	4.047e-2	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 (g=f(8,24), l=f(8,24), w=8)	2.934e-3	2.881e-4	7.472e-3	1.180e-3
DFE2 (g=f(6,15), l=f(5,12), w=16)	2.776e-3	2.707e-4	4.320e-2	2.443e-3
DFE3 (g=f(5,13), l=f(5,10), w=16)	3.267e-3	6.742e-4	7.289e-2	1.012e-2
DFE4 (g=f(5,13), l=f(5,10), w=24)	N/A	N/A	7.404e-2	1.005e-2

Precision Results

For $J = 64$ (which the CPU-simulations were run with) we have similar errors to those predicted by the CPU-simulations. For $J = 144$, we have slightly larger errors, but still acceptable for use.

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 (g=f(8,24), l=f(8,24), w=8)	2.934e-3	2.881e-4	7.472e-3	1.180e-3
DFE2 (g=f(6,15), l=f(5,12), w=16)	2.776e-3	2.707e-4	4.320e-2	2.443e-3
DFE3 (g=f(5,13), l=f(5,10), w=16)	3.267e-3	6.742e-4	7.289e-2	1.012e-2
DFE4 (g=f(5,13), l=f(5,10), w=24)	N/A	N/A	7.404e-2	1.005e-2

Performance (J=144, float system size: 227KiB - 453MiB)



[1] Maxeler MAX3A Vectis DFE (Xilinx Virtex6 SXT475 @ 150MHz).

[2] Dual-socket hyperthreaded Intel Xeon X5660s @ 2.67GHz.

Power Efficiency

We measure the power requirements of a software implementation on two different systems and compare against the most power efficient.

Build	Throughput (elements/s)	Power (W)	Efficiency (elements/J)	Relative Efficiency
System 1[2] (4-cores)	1.63e8	208[1]	7.83e5	0.972
System 2[3] (6-cores)	2.05e8	399[1]	5.14e5	0.638
System 2 (12-cores)	4.05e8	503[1]	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**

[1] Power consumption of unused Maxeler cards subtracted.
[2] Hyperthreaded Intel Core i7 870 @ 2.93GHz
[3] Dual-socket hyperthreaded Intel Xeon X5660s @ 2.67GHz.

Future work

- More complex models like shallow water.
- More sophisticated means for incorporating input-value uncertainty into a simulation.
- Generate designs from a domain-specific languages, allowing domain specialists to supply uncertainty information directly.

Conclusions

- Obtaining near-identical numerical behaviour to a reference solution is not always the best goal.
- Chaotic systems force us to re-evaluate how we measure the correctness of a simulation.
- Chaos-aware precision reduction enables us to exploit input-parameter uncertainty and nature of the chaotic behaviour itself.
- We demonstrated how to achieve this for simple chaotic system and the performance and power benefits over a conventional CPU implementation.
- Weather modelling is a problem that drives the purchase of supercomputers. Only reconfigurable computing enables us to exploit the properties of chaotic systems to reduce power and improve performance.