Area-Time-Precision on Demand

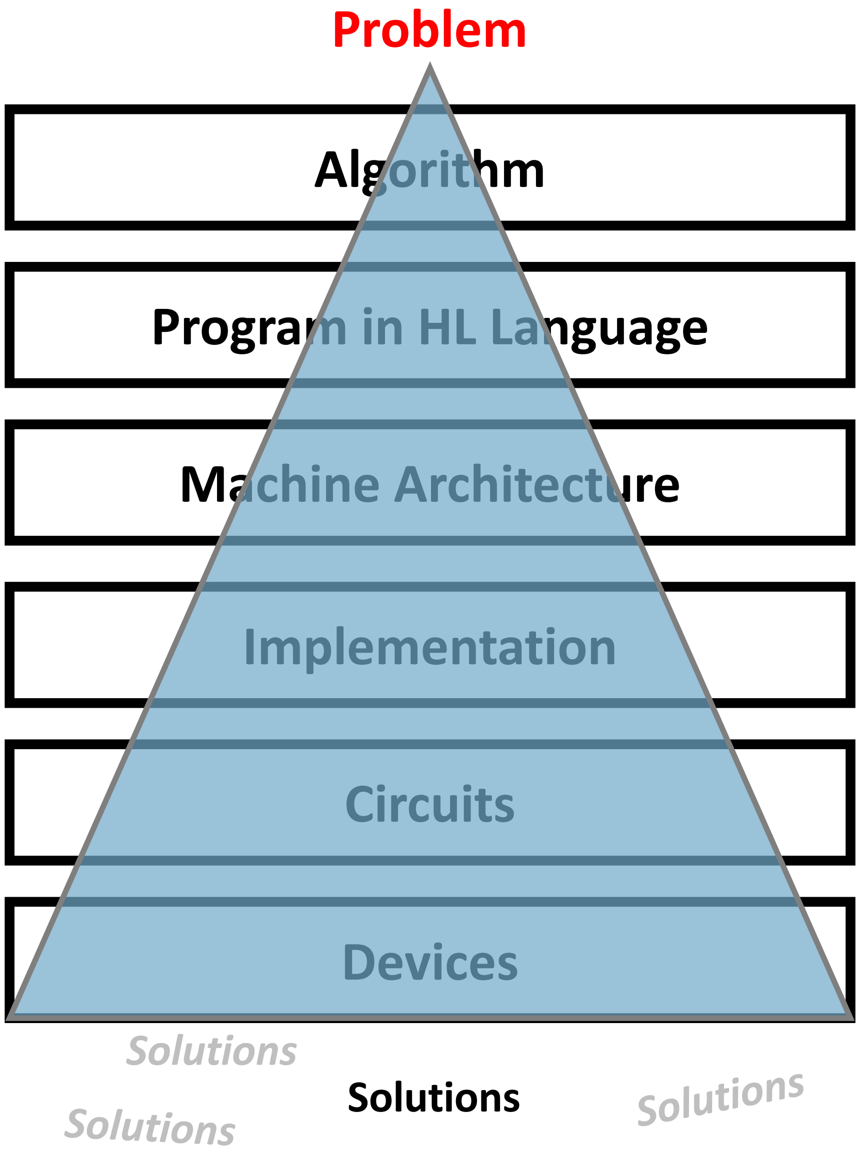
The Space-Time-Value Challenges of Reconfigurable Accelerator Design

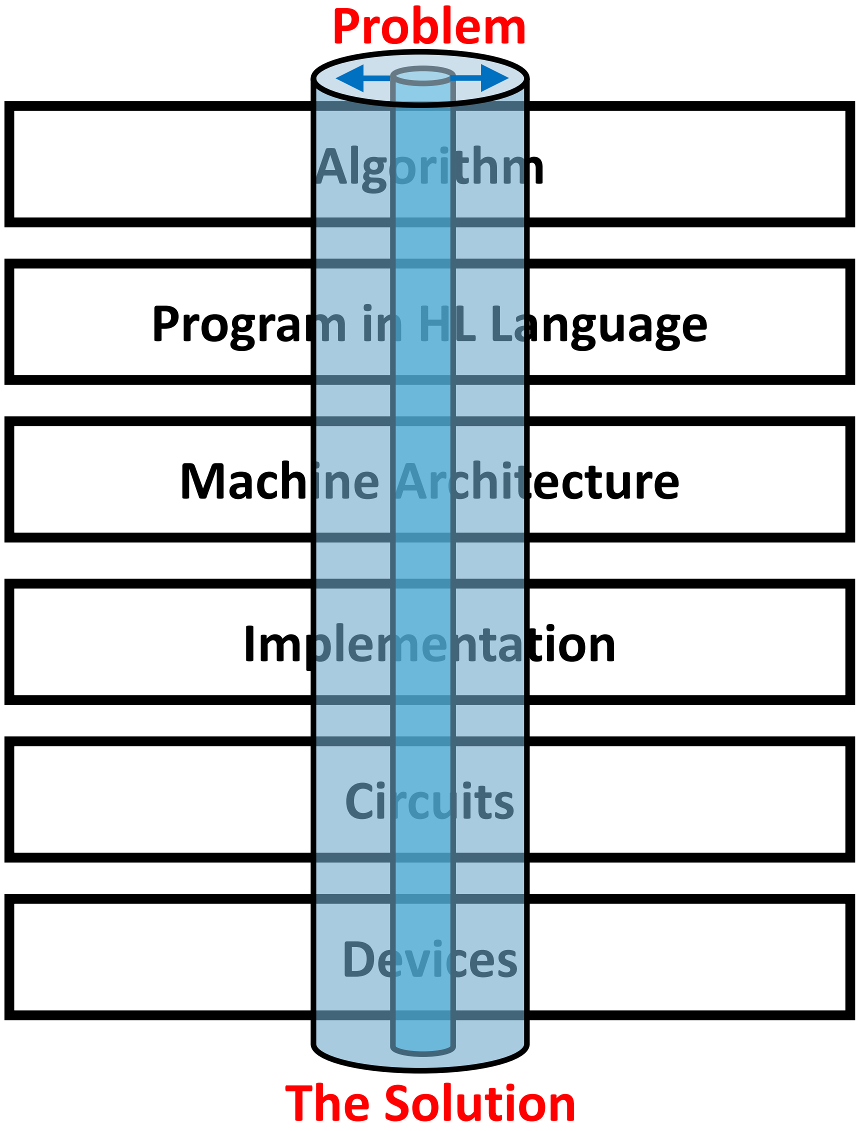
Georgi Gaydadjiev

Workshop on Reconfigurable Computing
January 17, 2024
Munich, Germany

TU Delft

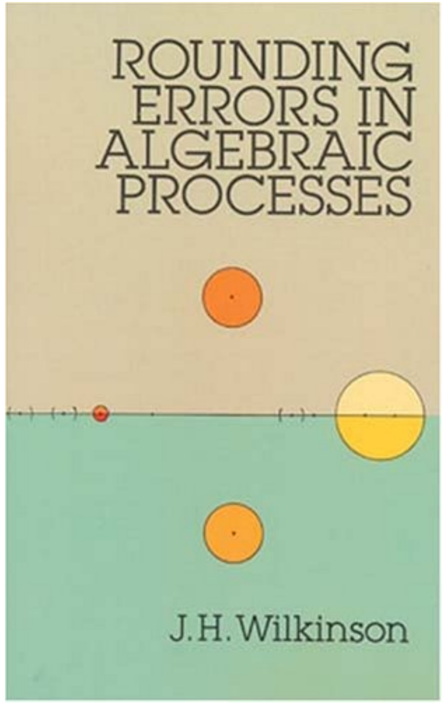
Thinking Vertically about Computing Problems





James Wilkinson on Value+Error

Computation can be described as
ideal infinite precision results + error
(J. Wilkinson)



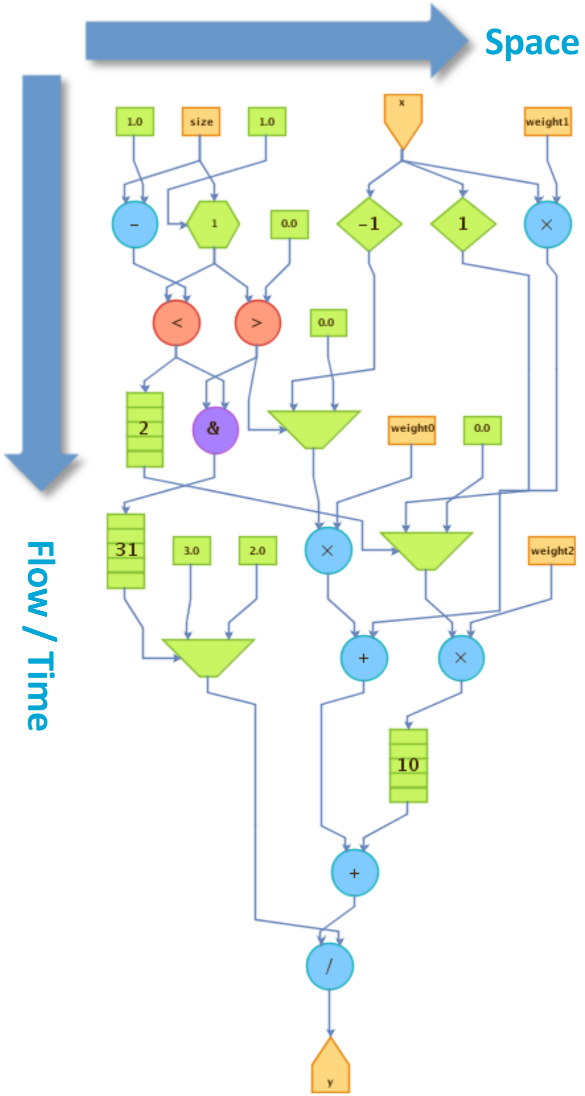
OR

as a multiscale discretization of

Space, Time and Value (STV)

Multiscale (Dataflow) Reconfigurable Computing enables the discretization of STV and direct tradeoff with performance, computational density, power consumption and total cost of computation.

Optimizations at all abstraction levels



Multiple scales of computing	Important features for optimization
complete system level	⇒ balance compute, storage and IO
parallel node level	⇒ maximize utilization of compute and interconnect
microarchitecture level	⇒ minimize data movements
arithmetic level	⇒ tradeoff range, precision and accuracy = discretize in Time, Space and Value
bit level	⇒ encode and add redundancy
transistor level	=> manipulate ‘0’ and ‘1’

and more, e.g., trade/hide Communication (Time) for/behind Computation (Space), etc

Easy it is not …

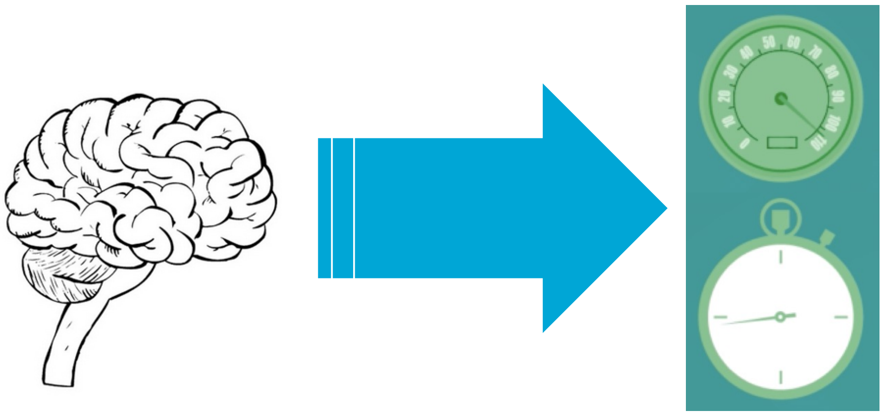
Easy it is not (and not really new)

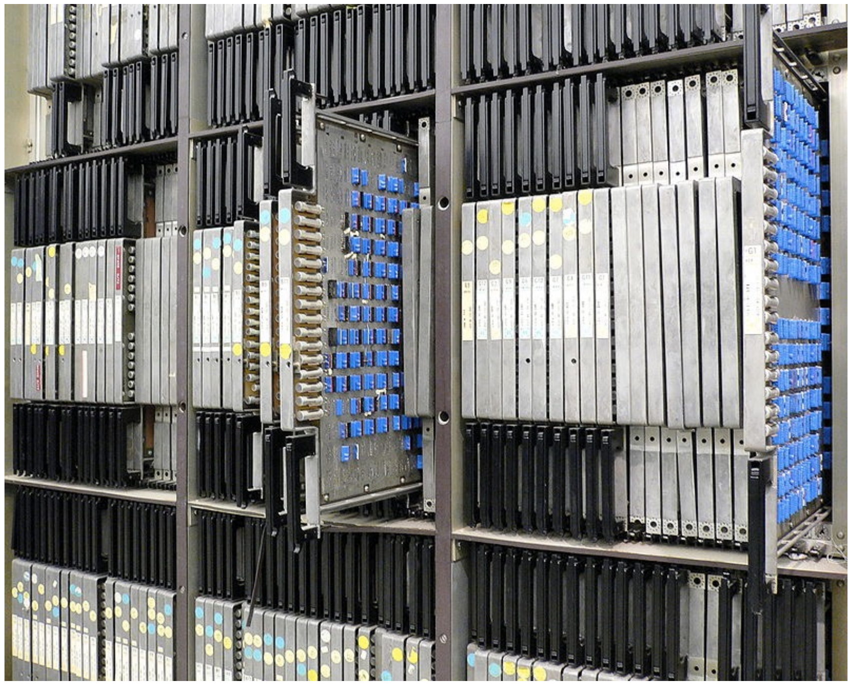
Slotnick's law (of effort):

"The parallel approach to computing does require that some ***original thinking*** be done **about numerical analysis and data management** in order to secure efficient use.

In an environment which has represented the absence of the need to think as the highest virtue this is a decided disadvantage."

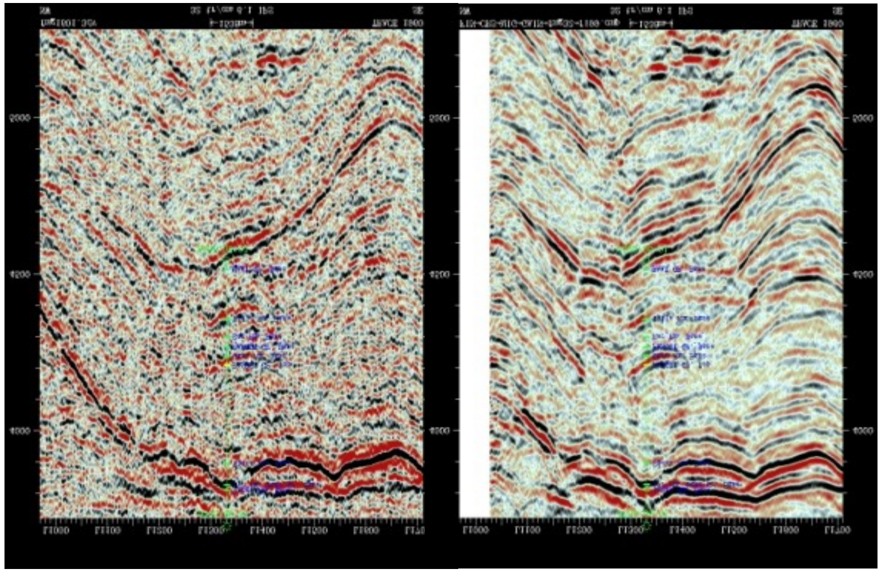
Daniel Slotnick (1931-1985)
Chief Architect of Illiac IV





Depends heavily on what is computed

Imaging: What does it mean for the result to be good enough?



?

IEEE Floating Point: Bit-accurate IEEE Floating Point, needed?

Accounting: Computing certain exact digits? Decimal? Binary?

Risk: Qualitative feedback might be enough? **1 bit:** will it rain or not?

Optimize representation for arithmetic and data movements

Floating Point

- Vary mantissa & exponent sizes
- Radix-4, radix 16, etc
- Block floating point
- Decimal floating point, etc

Advanced

- Logarithmic numbers
- Modulo Arithmetic
  (Chinese Remainder Theorem)
- Redundant Numbers

Integer

- Fixed Point
- Dual fixed point

Encode the wave field (STV):

- Predictive coding
- Arithmetic coding
- Lossless vs lossy
  - Wavelets
  - Curvelets, de-noising, etc

Limits on Computing + and ×

[Shmuel Winograd, 1965]

Bounds on Addition

- Binary: O(log n)
- Residue Number System: O(log 2log α(N))
- Redundant Number System: O(1)

Bounds on Multiplication

- Binary: O(log n)
- Residue Number System: O(log 2log β(N))
- Using Tables: O(2[log n/2]+2+[log 2n/2])
- Logarithmic Number System: O(Addition)

However, Binary and Log numbers are easy to compare, others are not!

Also, constant multiplication complexity depends on the number of ‘1’s ………

From ‘1’s to distance between ‘1’s

Rational Approximations & Continued Fractions

The M-log-Fraction

$$.001010011 = 2^{-\alpha_1} + 2^{-\alpha_1-\alpha_2} + 2^{-\alpha_1-\alpha_2-\alpha_3} + 2^{-\alpha_1-\alpha_2-\alpha_3-\alpha_4}$$

$$M_i = \alpha_i - M_{i-1}$$

is equivalent to

$$= \cfrac{1}{2^{M_1} + \cfrac{1}{-(2^{-M_1} + 2^{M_2}) + \cfrac{1}{(2^{-M_2} + 2^{M_3})\ddots}}} =$$

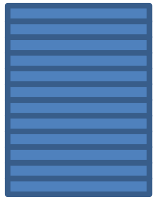
$$= [0; 2^{M_1}, -(2^{-M_1} + 2^{M_2}), (2^{-M_2} + 2^{M_3}), -(2^{-M_3} + 2^{M_4}), (2^{-M_4} + 2^{M_5})\ldots]$$

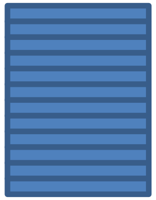
Oskar Mencer, Rational Arithmetic Units in Computer Systems
PhD Thesis, Stanford University, 2000.

Tradeoff compute versus memory

Computing f(x) in the range [a,b] with $|E| \leq 2^{-n}$

Table *Table+Arithmetic* *Arithmetic-only*



 and +,-,×,÷

+,-,×,÷

- uniform vs non-uniform
- number of table entries
- how many coefficients

- polynomial or rational approx
- continued fractions
- multi-partite tables

Underlying hardware/technology changes the optimum

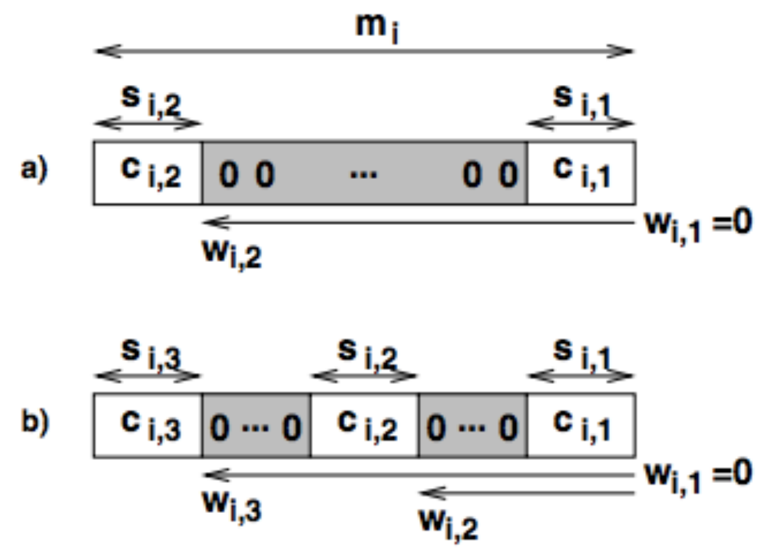
in Practice: Tradeoff Representation, Memory and Arithmetic

Minimal Latency (Optimized for Latency)

Range [bits] \ Precision [bits]	4	8	12	16	20	24
24	sin: *tp2*, 24, 78 log: *po2*, 8, 30 sqr: *tp2*, 18, 912	sin: *tp2*, 28, 348 log: *tp2*, 12, 78 sqr: *tp2*, 24, 2400	sin: *tp2*, 32, 792 log: *tp2*, 15, 384 sqr: *tp2*, 26,10368	sin: *tp2*, 32, 3168 log: *tp2*, 21, 1056 sqr: *tp2*, 30,23808	sin: *tp2*, 40, 7872 log: *tp2*, 24, 4800 sqr: *tp3*, 34,17920	sin: *tp3*, 42, 5504 log: *tp2*, 28,11136 sqr: *tp4*, 39,12800
20	sin: *po2*, 19, 61 log: *po2*, 7, 27 sqr: *tp2*, 18, 456	sin: *tp2*, 24, 300 log: *tp2*, 12, 78 sqr: *tp2*, 20, 2016	sin: *tp2*, 28, 696 log: *tp2*, 15, 384 sqr: *tp2*, 24, 4800	sin: *tp2*, 32, 3168 log: *tp2*, 21, 1056 sqr: *tp2*, 32,12288	sin: *tp2*, 32, 6336 log: *tp2*, 24, 4800 sqr: *tp3*, 33, 8704	sin: *tp3*, 38, 4992 log: *tp2*, 27,10752 sqr: *tp3*, 37,19456
16	sin: *tp2*, 16, 54 log: *po2*, 7, 27 sqr: *tp2*, 17, 324	sin: *tp2*, 19, 240 log: *tp2*, 12, 78 sqr: *tp2*, 18, 912	sin: *tp2*, 24, 600 log: *tp2*, 15, 384 sqr: *tp2*, 24, 2400	sin: *tp2*, 28, 2784 log: *tp2*, 21, 1056 sqr: *tp2*, 26,10368	sin: *tp2*, 32, 6336 log: *tp2*, 24, 4800 sqr: *tp2*, 30,23808	sin: *tp3*, 32, 4224 log: *tp2*, 27,10752 sqr: *tp3*, 34,17920
12	sin: *po2*, 9, 31 log: *po2*, 7, 27 sqr: *tp2*, 12, 156	sin: *tp2*, 16, 204 log: *tp2*, 12, 78 sqr: *tp2*, 18, 456	sin: *tp2*, 20, 504 log: *tp2*, 15, 384 sqr: *tp2*, 20, 2016	sin: *tp2*, 24, 2400 log: *tp2*, 21, 1056 sqr: *tp2*, 24, 4800	sin: *tp2*, 28, 5568 log: *tp2*, 24, 4800 sqr: *tp2*, 31,12288	sin: *tp2*, 32,12672 log: *tp2*, 27,10752 sqr: *tp3*, 33, 8704
8	sin: *po2*, 6, 22 log: *po2*, 7, 27 sqr: *tp2*, 7, 27	sin: *tp2*, 10, 132 log: *tp2*, 11, 72 sqr: *tp2*, 17, 324	sin: *tp2*, 16, 408 log: *tp2*, 15, 384 sqr: *tp2*, 18, 912	sin: *tp2*, 19, 1920 log: *tp2*, 21, 1056 sqr: *tp2*, 24, 2400	sin: *tp2*, 24, 4800 log: *tp2*, 24, 4800 sqr: *tp2*, 26,10368	sin: *tp2*, 28,11136 log: *tp2*, 27,10752 sqr: *tp2*, 30,23808
4	sin: *po2*, 6, 22 log: *po2*, 7, 27 sqr: *po2*, 7, 25	sin: *tp2*, 10, 132 log: *tp2*, 11, 72 sqr: *tp2*, 12, 156	sin: *tp2*, 15, 384 log: *tp2*, 15, 384 sqr: *tp2*, 18, 456	sin: *tp2*, 19, 1920 log: *tp2*, 17, 1056 sqr: *tp2*, 20, 2016	sin: *tp2*, 23, 4608 log: *tp2*, 24, 4800 sqr: *tp2*, 24, 4800	sin: *tp2*, 28,11136 log: *tp2*, 27,10752 sqr: *tp2*, 31,12288

Dong-U Lee, et.al.,Optimizing Hardware Function Evaluation, *IEEE Transactions on Computers*. vol. 54, no. 12, pp. 1520-1531. Dec, 2005

Next: Minimize ‘1’s => Sparse Coefficients



$$p = \frac{32799}{32768} - \frac{609}{32768}x - \frac{14881}{32768}x^2.$$

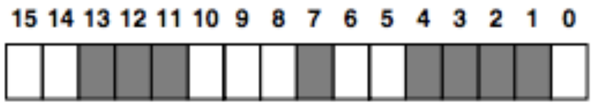


Fig. 2. Target format for cos function.

$$p = -\frac{75}{32768} + \frac{34538}{32768}x - \frac{6169}{32768}x^2.$$

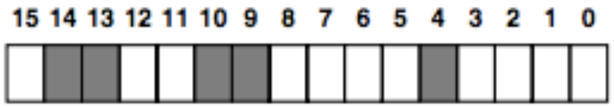


Fig. 3. Target format for sin function.

$$p = \frac{32793}{32768} + \frac{31836}{32768}x + \frac{21146}{32768}x^2.$$

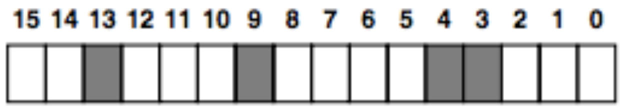


Fig. 4. Target format for exp function.

Nicolas Brisebarre, Jean-Michel Muller and Arnaud Tisserand
Sparse Coefficient Polynomial Approximations for Hardware Implementation,
Asilomar Conference, 2004.

12

Coefficients, Coefficients, FD Coefficients…

3D Finite Difference Coefficients

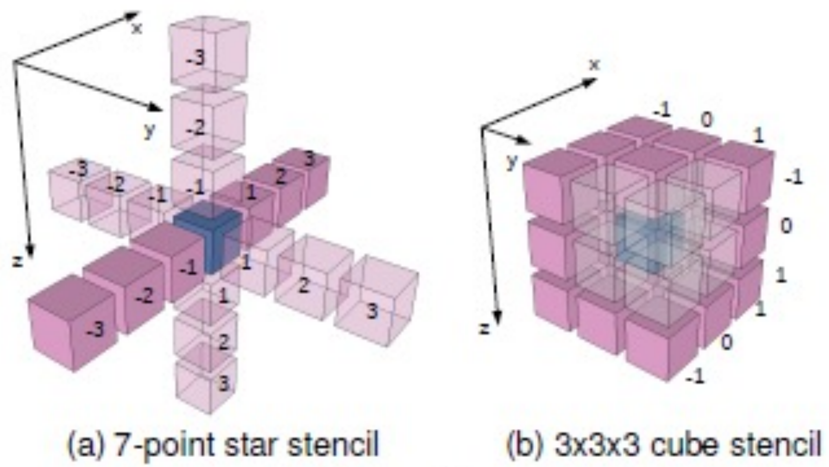
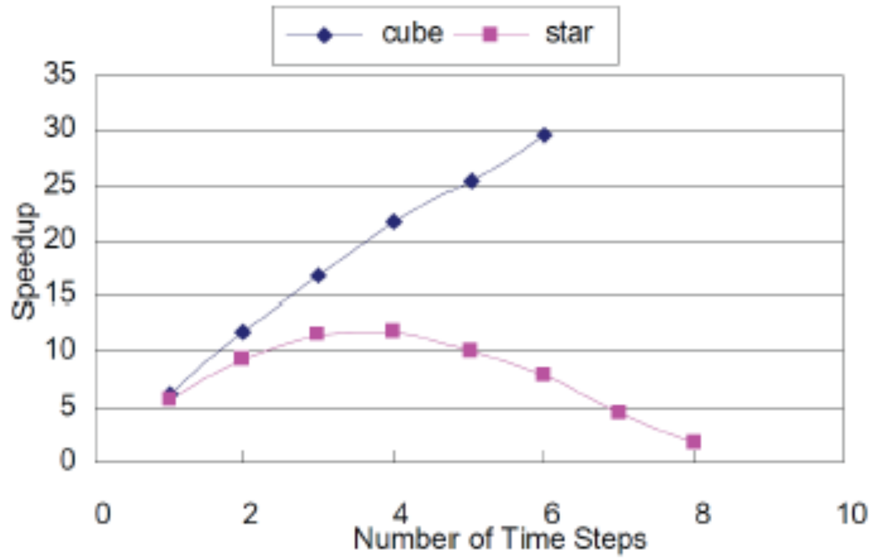


Figure 1: 2 Alternative stencil choices.

19 MADDs　　27 MADDs

Local Buffer = 6 slices　　Local Buffer = 3 slices

Time to compute is consequence of distance of coefficients in memory



Local temporal parallelism
=> Cascading timesteps

Motivation for Elementary Functions

- Used by compute intensive applications
- Evaluating on CPU
  - Many cycles in software or CPU microcode
- Evaluating on Reconfigurable HW (FPGA):
  - Evaluation stages add latency
  - Resources reduce to required precision and bit width
- Function composition at the cost of just one function
  - **Similar costs** in terms of hardware resources for *approximating* the value at a given x for expressions like $f(x) = \log(1+ \exp(-x^2/2))$ and $g(x) = \exp(x)$.

Resource utilisation: floating point[1]

	LUT	FF	BRAM	DSP	LUT	FF	BRAM	DSP
	dfeFloat(8,24)				**dfeFloat(11,53)**			
multiplication	155	364	2	1	343	696	3	4
addition	582	657	4	0	1,025	1,307	2	0
division	3,302	3,188	10	0	9,713	7,881	24	0
sqrt	470	897	1	0	1,741	3,356	1	0
sin	679	1,082	7	4	2,053	3,928	28	16
cos	693	1,072	7	4	2,082	3,908	28	16
exp	781	1,201	3	5	2,495	3,759	6	22
pow2	684	961	3	3	2,097	3,131	6	14
log2	541	916	5	4	1,533	3,163	26	16

[1] Maia DFE, pipelining 1.0. Numbers may differ for each compilation. Sampled with MaxCompiler 2016.1.1

Resource utilisation: fixed point[2]

	LUT	FF	BRAM	DSP	LUT	FF	BRAM	DSP
	dfeFix(16,16, TWOSCOMPLEMENT)				dfeFix(32,32, TWOSCOMPLEMENT)			
multiplication	36	33	0	2	193	536	0	8
addition	16	17	0	0	32	33	0	0
division	1210	2,378	0	0	4,786	8,509	19	0
sqrt	347	352	0	0	1,271	1,275	0	0
sin	349	431	4	8	1,255	2,909	26	26
cos	365	448	4	8	1,287	2,947	26	26
exp	1053	1,485	4	8	1,699	2,920	5	16
pow2	904	1,198	1	4	1,322	1,868	1	7
log	248	317	2	5	659	1,193	4	14

[2] Maia DFE, pipelining 1.0. Numbers may differ for each compilation. Sampled with MaxCompiler 2016.1.1

What about custom functions

- Function value is *approximated* at a given point
  - vast literature on function approximations in hardware
  - libraries: CPU (e.g. fdlibm), FPGA (e.g. FloPoCo), …
  - multi-precision approximations: Maple, Mathematica, etc.
- Options in hardware:
  - Simple lookup table
  - Iterative methods (e.g., Newton-Raphson)
  - Approximations on one interval [a,b]
  - Piece-wise approximations on many subintervals
  - Combination of lookup tables and shifts
  - Various combinations of all above

Small lookup table example

Problem: implement f(x) = sin(x) for 12 bit fixed point x

- There are 2^{12} = 4,096 function values in total
- Each value is only 12 bit wide
- hardware implementation as a lookup into FMEM
  Cost: no more than 4 BRAMs

A bit of CPU work:

- tabulate the function on CPU
- define mapped ROM

Iterative methods

- E.g., Newton-Raphson $\varphi(x) = 0 \quad \Rightarrow \quad x_{n+1} = x_n - \frac{\varphi(x_n)}{\varphi'(x_n)}$
- Works well when rhs simplifies to a polynomial
- Example: evaluate $f(x) = \frac{1}{\sqrt{x}}$ at $x = a.$

  Choose $\varphi(x) = \frac{1}{x^2} - a$ => $x_{n+1} = \frac{x_n}{2}\left(3 - a x_n^2\right)$
- Notes
  - Needs differentiable $\varphi(x)$, converges to a local minimum
  - sensitive to initial guess
  - quadratic convergence: precision roughly doubles

    => you can start iterations in small bit width

Approximations on one interval

- 3 steps to compute f(x)
  - Step 1: Argument Reduction = **g(x)** (bare for the next slide)
  - Step 2: Approximation of **g(x)** over interval [a,b]
    1. Lookup Table for a small number of bits
    2. Lookup Table + Add/Sub => Bi-partite tables
    3. Lookup Table + Mult-Add => Piecewise Linear Approx
    4. Shift-and-Add Methods => e.g., CORDIC
    5. Polynomial and Rational Approximations
    6. Almost never use Taylor series: converges slowly!
  - Step 3: Reconstruction to original argument (if necessary)

Simple argument reduction

- Function is periodic: can shift x towards the origin
- Example: sin(float x)

```
float sin(float x){
    float y = x mod (π/2);   // argument reduction
    float r1 = c0*y*y+c1*y+c2;
    float r2 = c3*y*y+c4*y+c5;
    return (r1/r2);          // rational approximation
}
```

- c_0-c_5 are coefficients of a rational approximation of sin(x) in [0, π/2]
- How to generate coefficients c_0-c_5. Use computer algebra system: Wolfram alpha (Mathematica), Maple,...

More complicated argument reduction

- Function y = exp(x). Reducing x to r in [-ln(2)/2, +ln(2)/2]:
  - Find integer N such that r := (x – N*ln(2))/2 is in the interval
  - Equivalently, x = N (0.5 ln 2) + r
  - Using identities: exp(x) := $2^{0.5N}$ exp(r)
- Step 1:
  - N := integer quotient of x/(0.5 ln 2). Adjust N to make it even!
  - calculate r as accurate as you can
- Step 2:
  - Compute exp(r) by approximation (e.g. polynomial)
  - Inaccurate r yields inaccurate exp(r)...
- Step 3:
  - Compute exp(x) = $2^{0.5N}$ exp(r) = 2^{k} exp(r) -- just a shift! If N=2k

Evaluating Polynomials

$$f(x) \approx \cdots + c_3 x^3 + c_2 x^2 + c_1 x + c_0$$
$$= (((\cdots + c_3)x + c_2) \cdot x + c_1) \cdot x + c_0$$

- **Horner Rule** transforms polynomial into a "Multiply-Add Structure"
- Multiply-Add is more numerically stable
- Multiply-Add takes less HW resources than multiply and add as 2 separate operations

Piece-wise approximations

- Many approximations locally defined on their sub-intervals [ai, bi].
- Approximations only differ by e.g., polynomial coefficients
- For every x find its interval
- Table lookup: get coefficients for this interval
- Evaluate e.g., polynomial
- Does not hurt to employ argument reduction: less intervals, higher convergence in each interval
- How to generate: use compute algebra system. Remez method (minimax polynomial), splines...

Further Reading on Function Evaluation

- **J.M. Muller, "Elementary Functions," Birkhaeuser, Boston, 1997.**
- Story, S. and Tang, P.T.P., "New algorithms for improved transcendental functions on IA-64," in Proceedings of 14th IEEE symposium on computer arithmetic, IEEE Computer Society Press, 1999.
- D.E. Knuth, "The Art of Computer Programming", Vol 2, Seminumerical Algorithms, Addison-Wesley, Reading, Mass., 1969.
- C.T. Fike, "Computer evaluation of mathematical functions," Englewood Cliffs, N.J., Prentice-Hall, 1968.
- L.A. Lyusternik, "Handbook for computing elementary functions", available in English translation.

Euclids Elements, Representing $a^2+b^2=c^2$
=> optimal representation is important

Maximum Performance Computing
=> Kolmogorov Complexity (K)

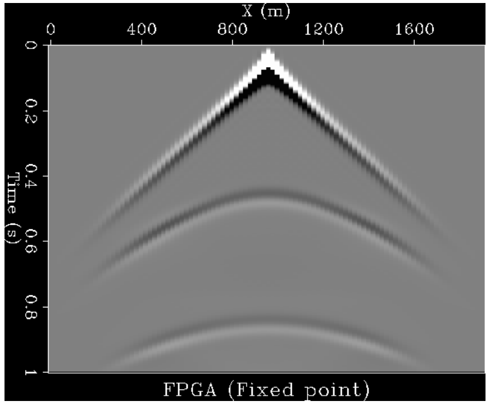
Definition (Kolmogorov*):
"If a description of *string s*, *d*(*s*), is of minimal length, […]
it is called a **minimal description** of *s*. Then the length of *d*(*s*), […] is the **Kolmogorov complexity** of *s*, written *K*(*s*), where *K(s)* = |*d(s)*|"

Of course K(s) depends heavily on the Language L used to describe actions in K. (e.g. Java, Esperanto, an Executable file, etc)

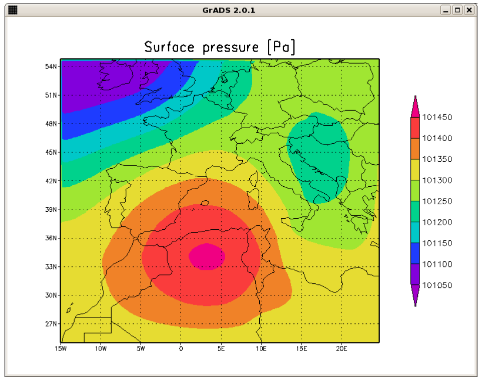
Optimal Representation is a hard problem ontop of a hard problem.

*Kolmogorov, A.N. (1965). "Three Approaches to the Quantitative Definition of Information". *Problems Inform. Transmission* **1** (1): 1–7.

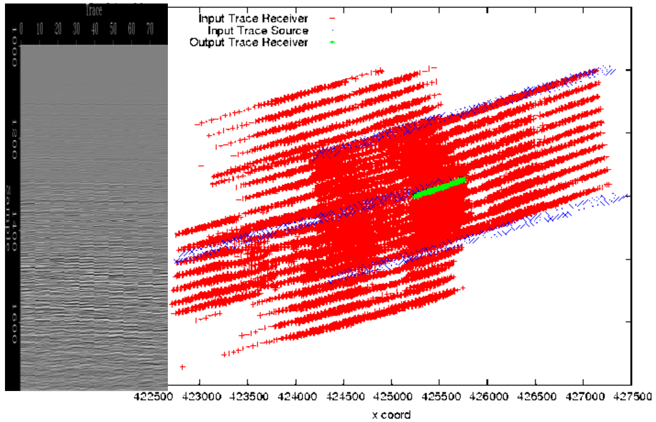
Comparing an x86 based 1U machine with a Multiscale Dataflow based 1U machine with 8 DFEs



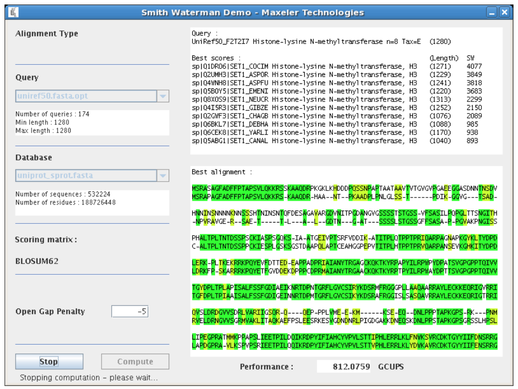
Modelling 25x



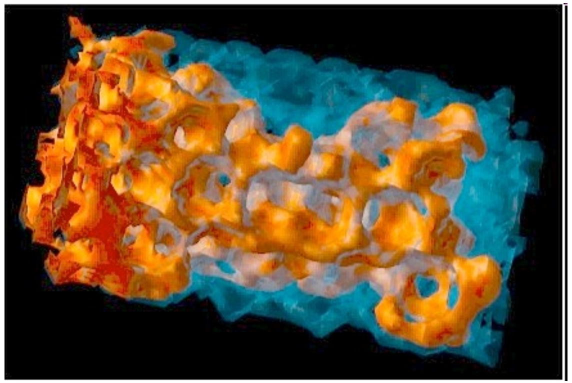
Finite Difference 60x



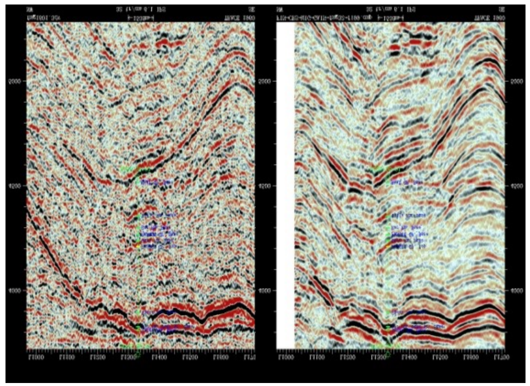
Data Correlation 22x



Smith-Waterman 16-32x

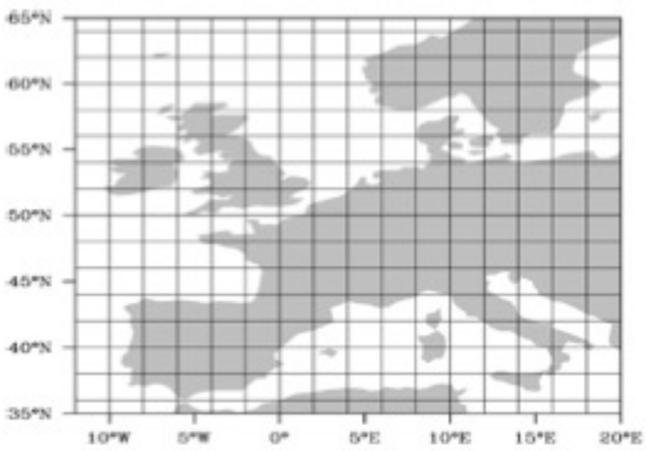


Fluid Flow 30x

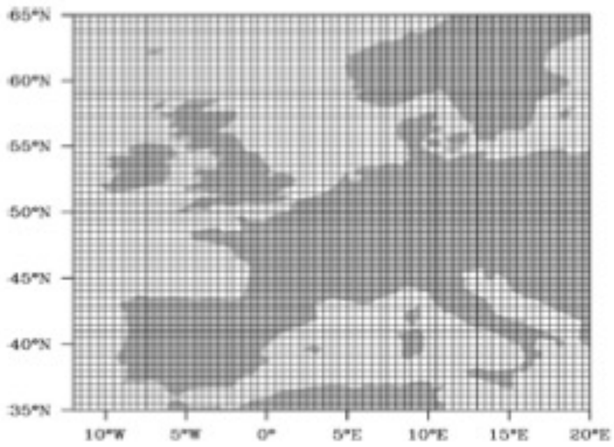


Imaging 29x

Weather and climate models on DFEs



Which one is better?

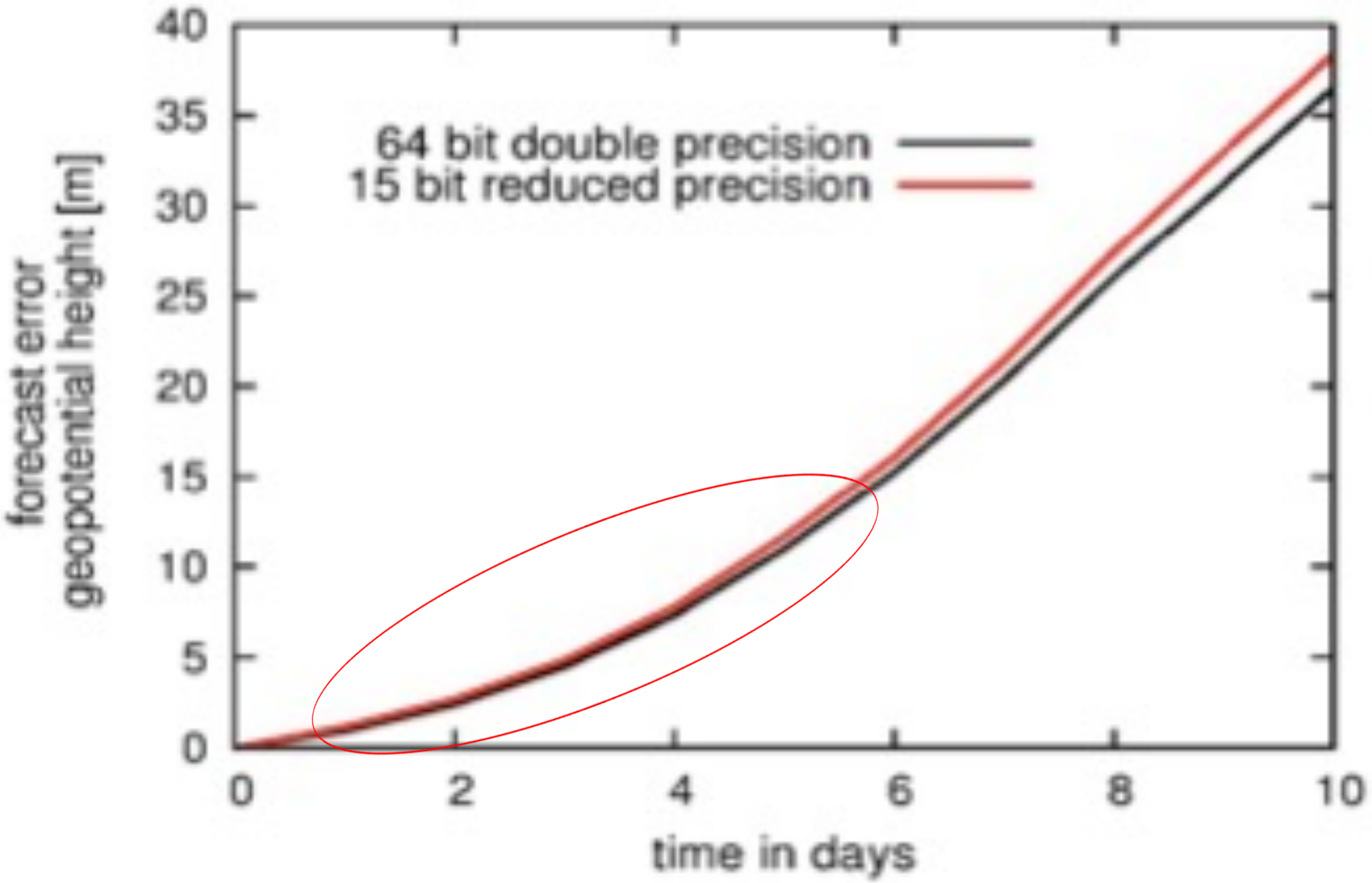


Finer grid and higher precision are obviously preferred but the computational requirements will increase ➔ Power usage → $$

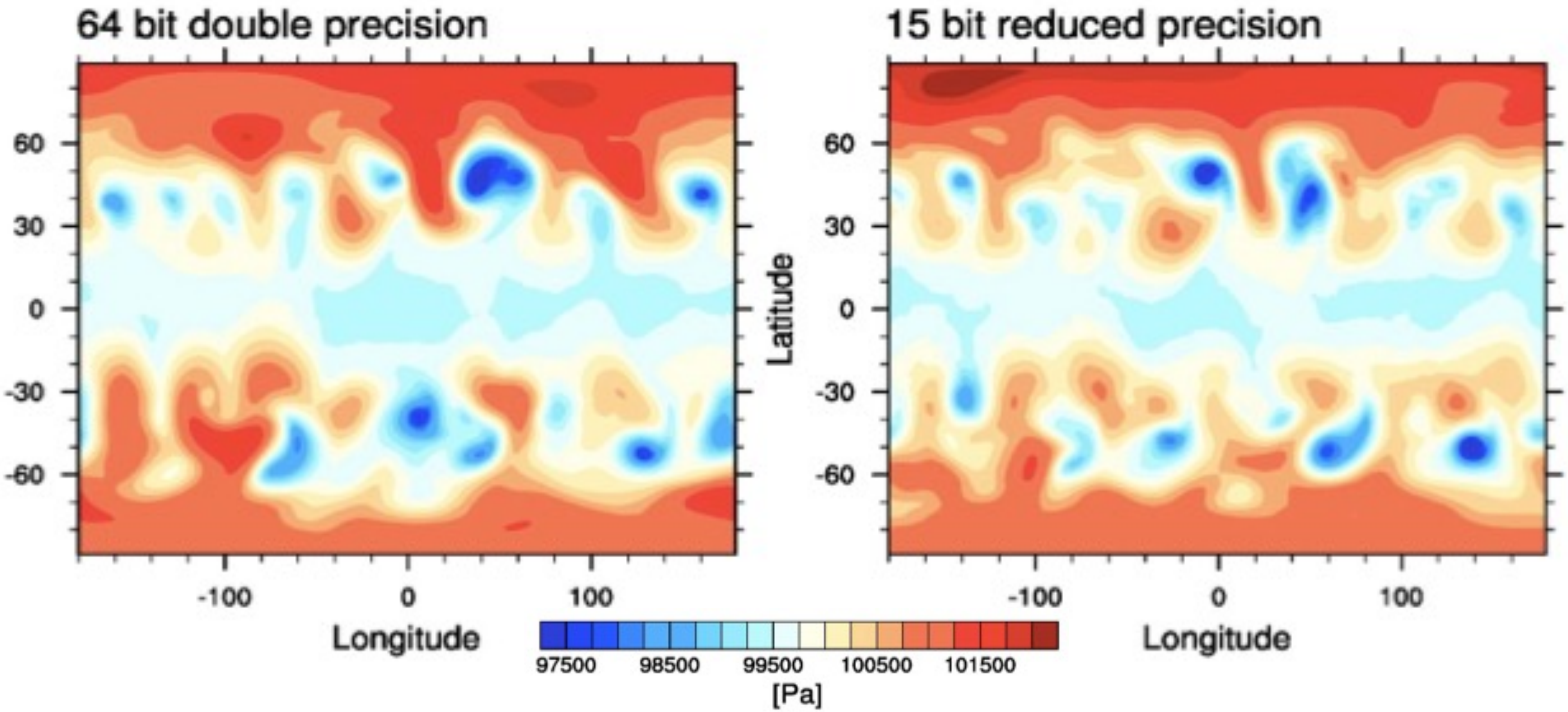
What about using reduced precision? (15 bits instead of 64bit double precision FP)

We use only **15 bits** for 98% of the computation:

Weather models precision comparison



What about 15 days of simulation?



Surface pressure after **15 days** of simulation for the double precision and the reduced precision simulations (quality of the simulation hardly reduced)

Concluding remarks

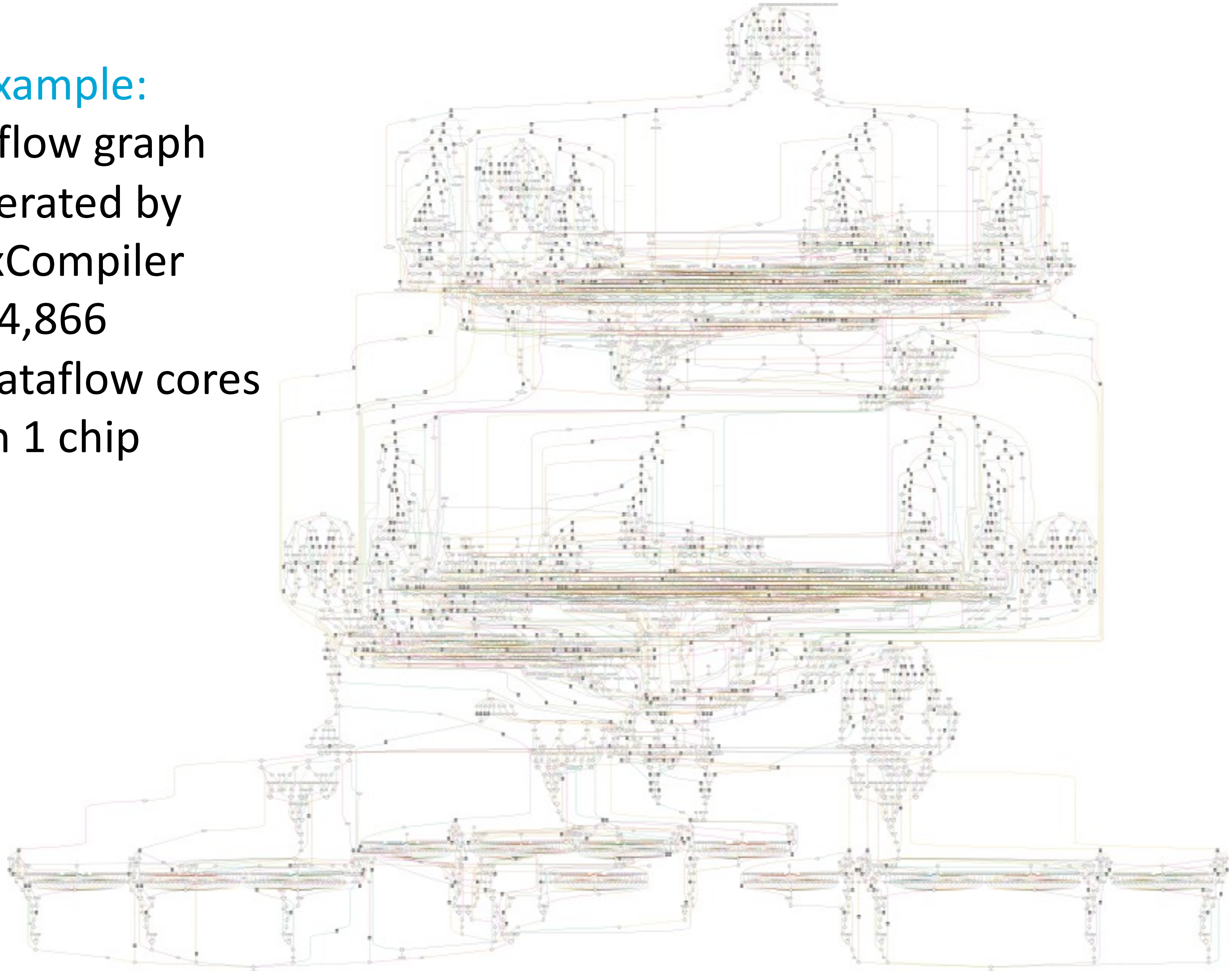
- Reconfigurable acceleration success points to the weakness of evolutionary approaches to parallel processing: hardware (multi core) and software (C++, etc.), at least for HPC applications
- The automation of acceleration is still early on; still required: tools, methodology for writing apps., analysis methodologies, and
- a new hardware substrate (coarser grain, higher speed, shorter P&R times) maybe DFRA (way too many letters)

Concluding remarks (2)

- Reconfigurable HPC can become a reality if underlying software problems can be solved
- In HPC the parallel approach demands rethinking algorithms, data representation, programming approach, models of computation and environment (and hardware)
- There's a lot of research ahead to effectively create parallel translation and array based technology
  - How much automation?
- Tools, Tools, Tools + dedicated methodologies

Example:

data flow graph
generated by
MaxCompiler
4,866
static dataflow cores
in 1 chip



Doable with VHDL / Verilog?
Also for nuclear physicists?

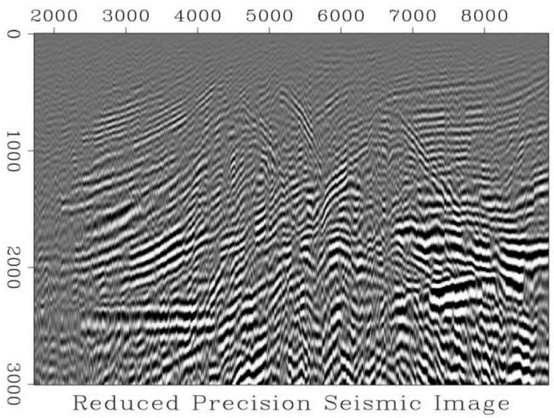
Thank you very much for your attention

?

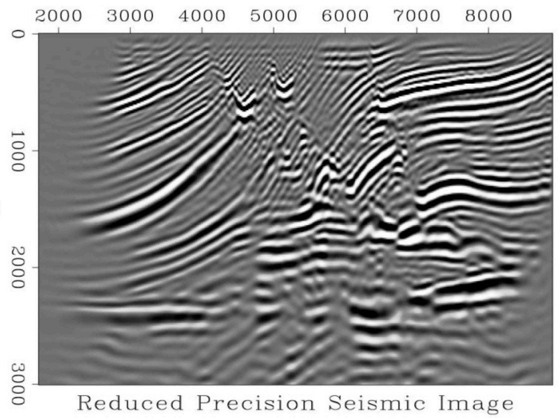
35

Fixed-point bit-width exploration

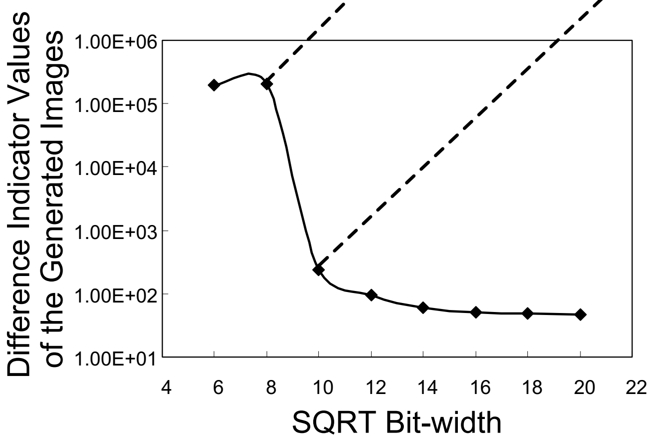
8-bit fixed-point



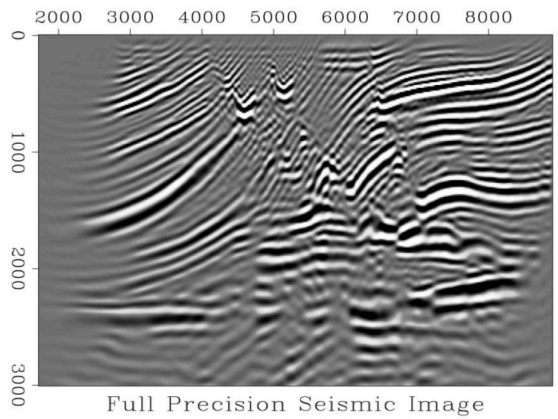
10-bit fixed-point



Different parts are explored *separately*, i.e., when we investigate one part, we keep the bit-widths in other parts a constant high value



'true' image: single-precision floating-point

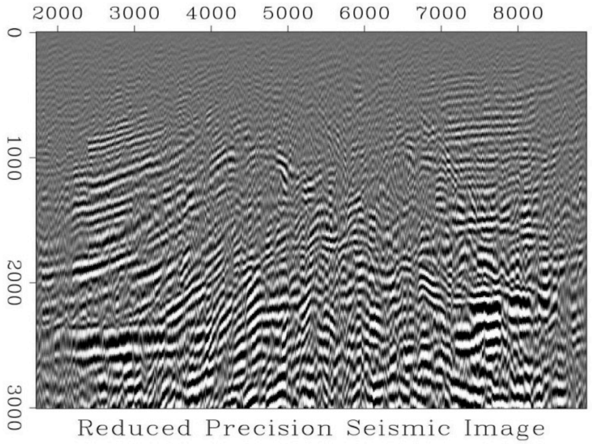


Similarly, we observe a significant drop of the error when the SQRT bit-width increases from 8 to 10

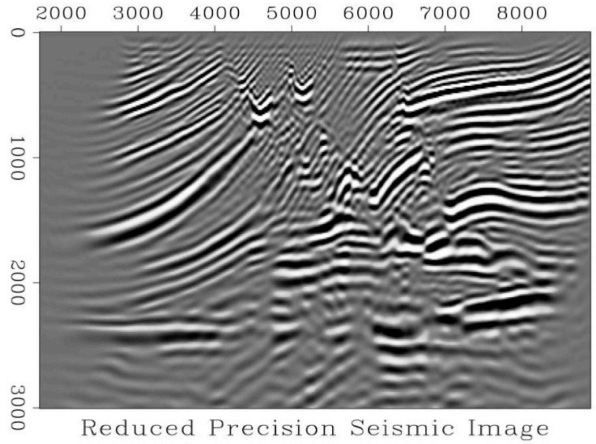
Similar precision thresholds observed in both synthetic and field results. This behavior enables an automatic tool to determine the minimum precision that still keeps the result *good enough*

Floating-point bit-width exploration

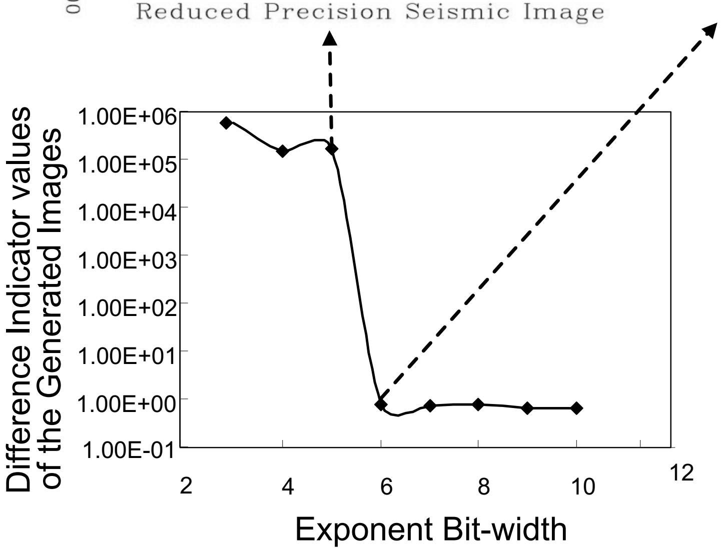
floating-point: 5-bit exponent



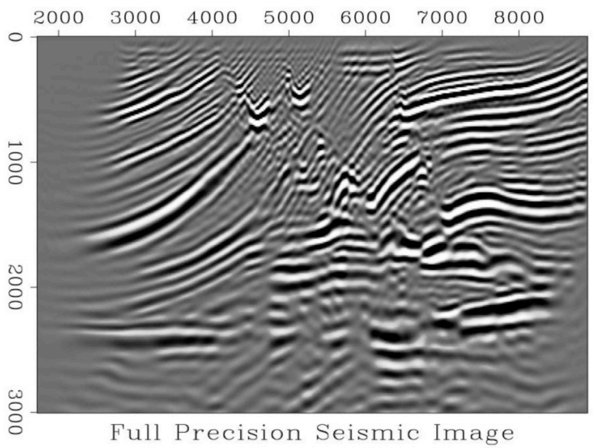
floating-point: 6-bit exponent



We use the Marmousi synthetic data set as the test data, and explore different combinations of exponent and mantissa bit-width



'true' image: single-precision floating-point

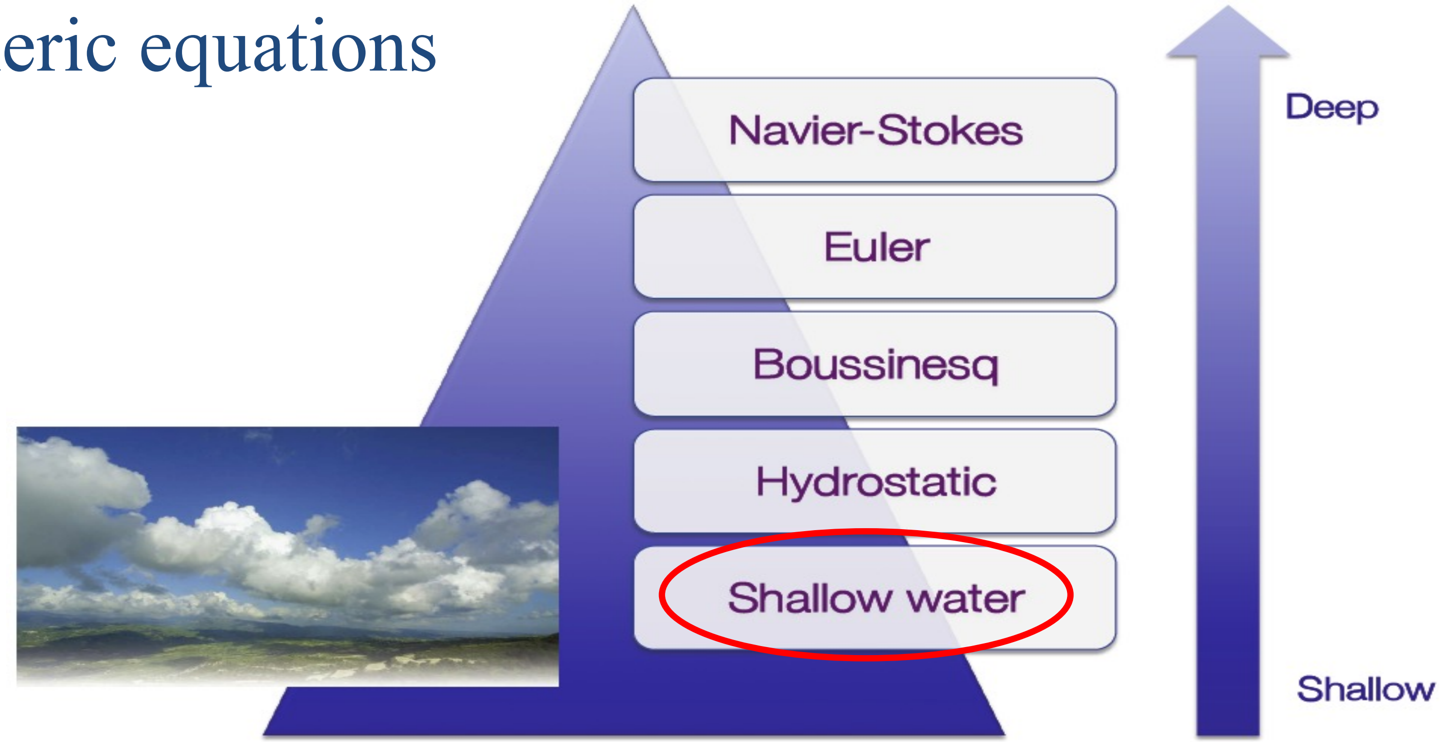


A precision threshold at exponent width of 6 bits:

- The error drops significantly when we increase the exponent width from 5 bits to 6 bits
- The image also turns from nearly random noise at 5 bits, to almost identical to the 32-bit image at 6 bits

Global Weather Simulation (10 years ago!)

- Atmospheric equations



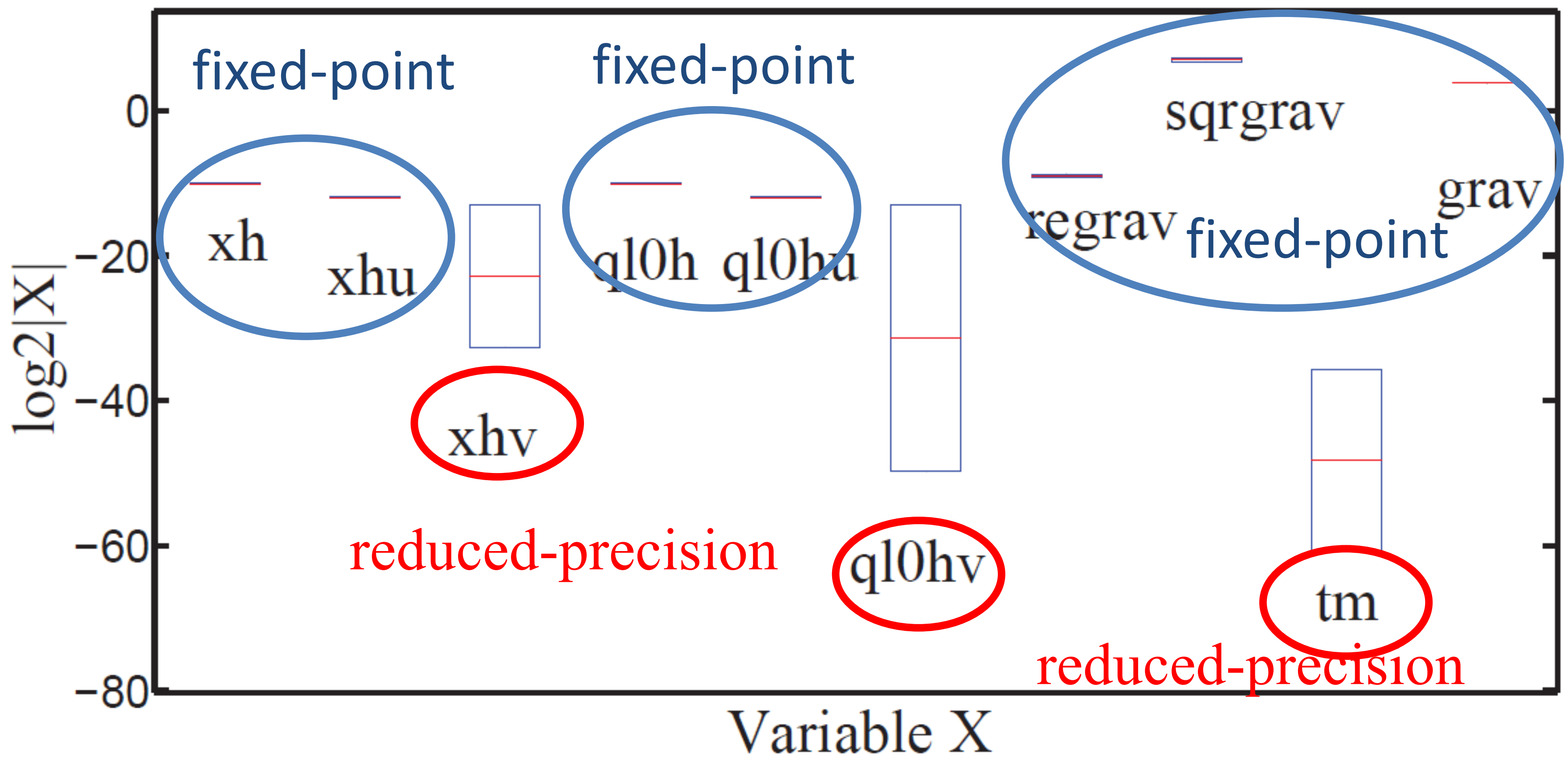
- Equations: Shallow Water Equations (SWEs)

$$\frac{\partial Q}{\partial t}+\frac{1}{\Lambda}\frac{\partial(\Lambda F^{1})}{\partial x^{1}}+\frac{1}{\Lambda}\frac{\partial(\Lambda F^{1})}{\partial x^{2}}+S=0$$

[L. Gan, H. Fu, W. Luk, C. Yang, W. Xue, X. Huang, Y. Zhang, and G. Yang, Accelerating solvers for global atmospheric equations through mixed-precision data flow engine, FPL2013]

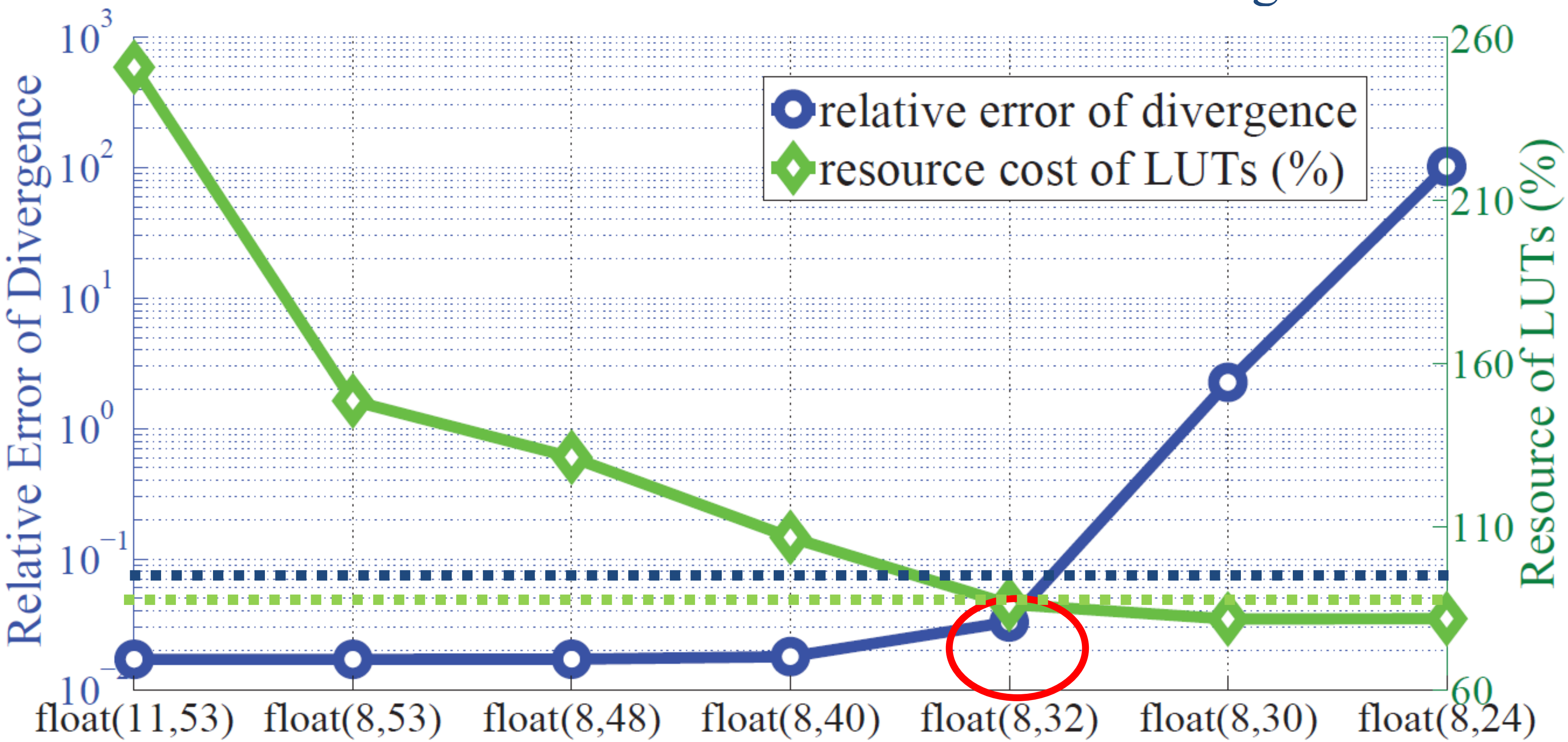
Always double-precision needed?

- Range analysis to track the absolute values of all variables



What about error vs area tradeoffs

- Bit accurate simulations for different bit-width configurations.



Accuracy validation

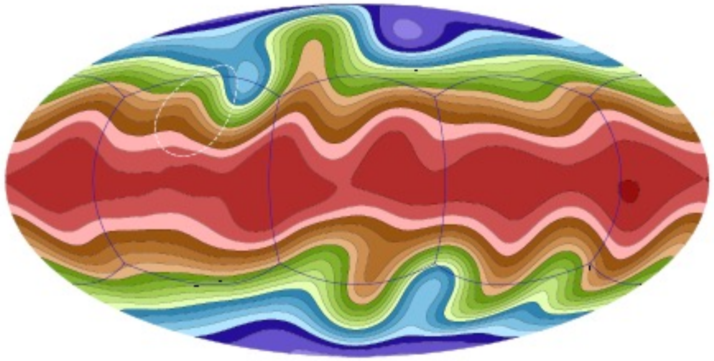


Figure 15. Surface level distribution of the atmosphere at day 15 in the isolated mountain test. Results are obtained on a 10,240 10,240 6 cubed-sphere mesh using 1,536 nodes of the Tianhe-1A. The conical mountain is outlined by the dotted circle in the figure.

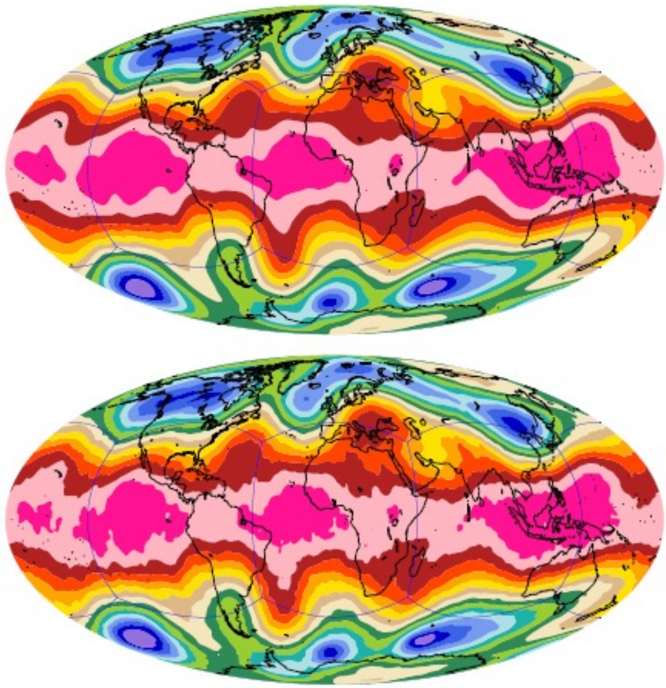


Figure 16. Surface level distribution of the atmosphere at day 15 in the real-topography test. We compare results at a 40-km resolution (upper panel) and a 1-km resolution (lower panel).

[Chao Yang, Wei Xue, Haohuan Fu, Lin Gan, et al. 'A Peta-scalable CPU-GPU Algorithm for Global Atmospheric Simulations', PPoPP'2013]

And there is also performance gain

Platform	Performance ()	Speedup
6-core CPU	4.66K	1
Tianhe-1A node	110.38K	23x
MaxWorkstation	468.1K	**100x**
MaxNode	1.54M	**330x**

14x

Meshsize: 1,024×1,024×6

MaxNode speedup over Tianhe node: 14 times

Imperial College London

ISCAS

High Performance Computing

TU Delft

And power efficiency too

Platform	Efficiency ( )	Speedup
6-core CPU	20.71	1
Tianhe-1A node	306.6	14.8x
MaxWorkstation	2.52K	**121.6x**
MaxNode	3K	**144.9x**

Meshsize: 1,024×1,024×6

9 x

MaxNode is 9 times more power efficient