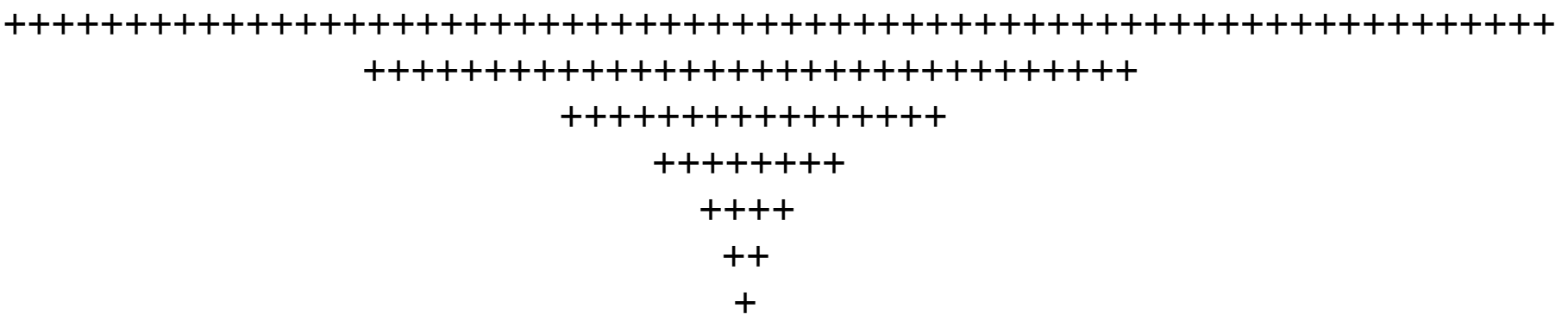
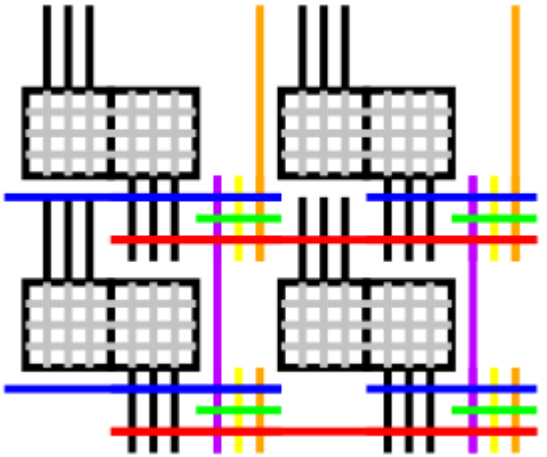
Accurate Parallel Floating-Point Accumulation



Edin Kadric Paul Gurniak André DeHon



What the paper is about

- Iterative algorithm to compute the ***accurate*** sum of ***Floating-Point*** numbers in ***parallel***
- Proof of convergence
- The algorithm is ***fast***
  - Only ***2*** iterations
- Efficient hardware implementation
  - For the algorithm
  - For a Double precision residue-preserving FP adder
    - ***Same speed*** as FP adder, only ***1.5x*** the area

Outline

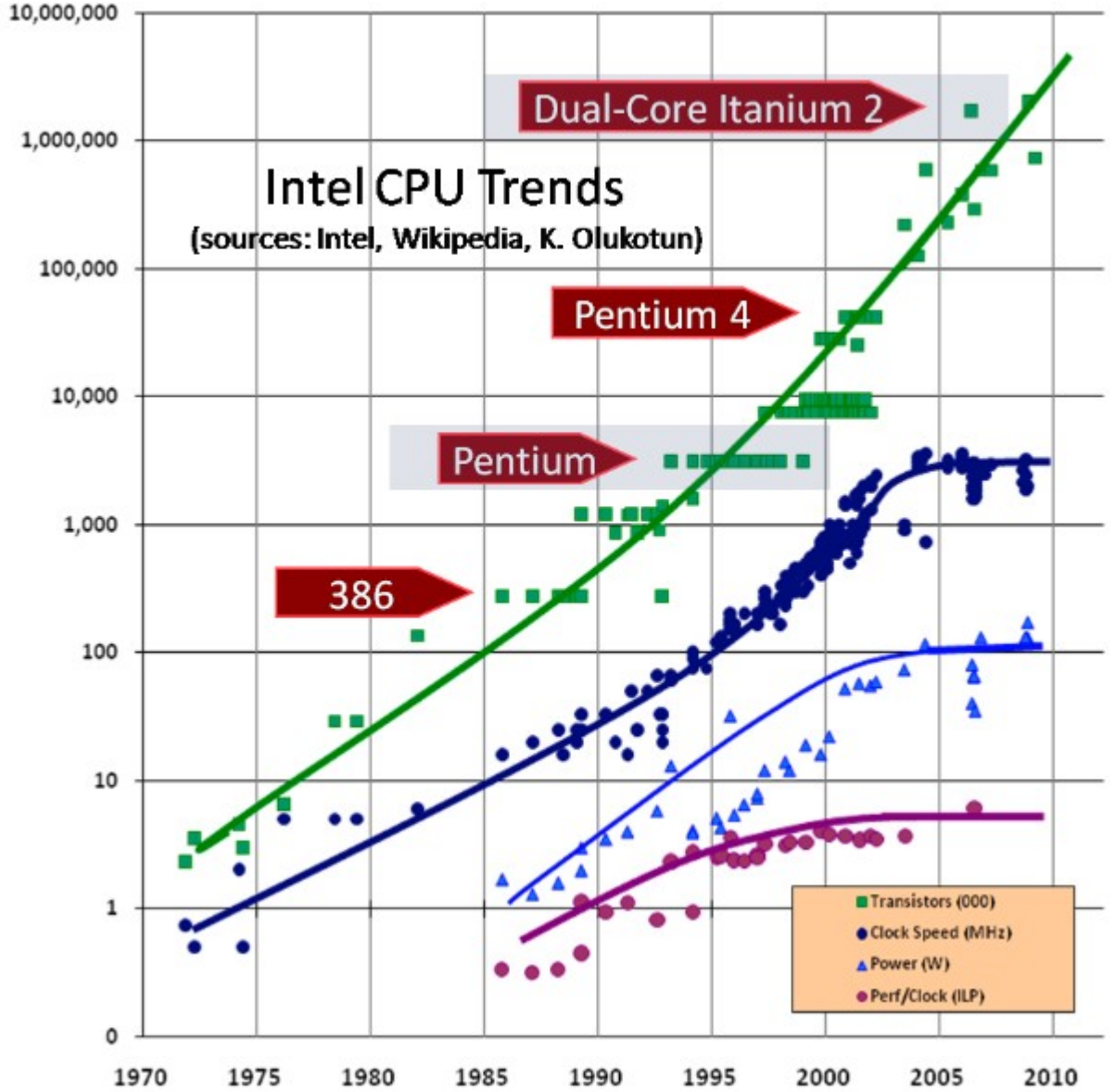
- Motivation
- Algorithm
- Simulation results
- Hardware Implementation
- Future work
- Conclusion

Outline

- **Motivation**
- Algorithm
- Simulation results
- Hardware Implementation
- Future work
- Conclusion

Motivation: How to speed up our circuits?

- Moore's law, more transistors, stagnated clocks
- Speed-up if parallelism extracted

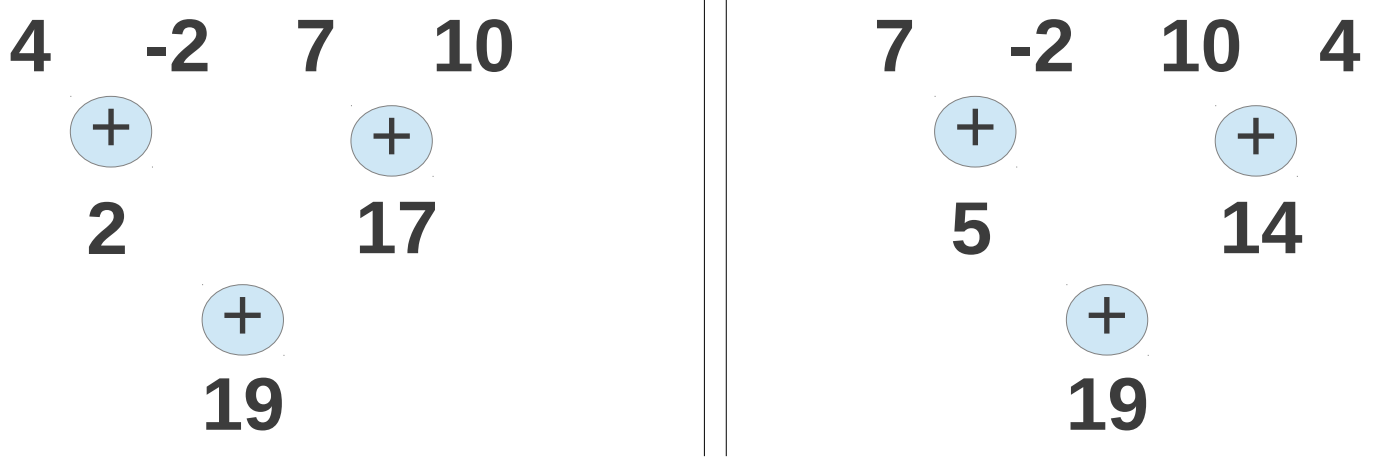


http://www.gotw.ca/publications/concurrency-ddj.htm

Motivation

Use more transistors to speed up arithmetic

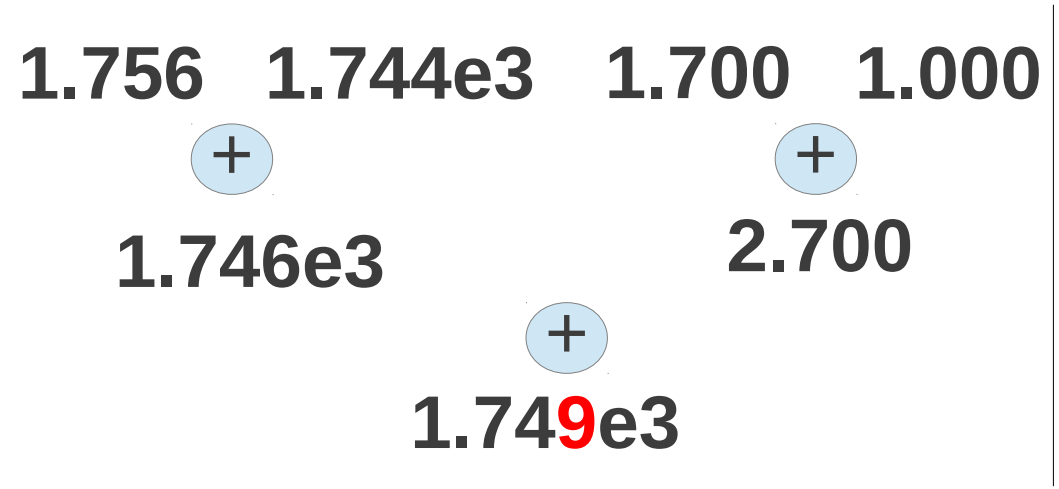
- Integer Addition:
  - Task: Want to add N numbers together
  - Solution: Build a binary tree of depth log(N)
    - Works because addition is associative: A+(B+C)=(A+B)+C

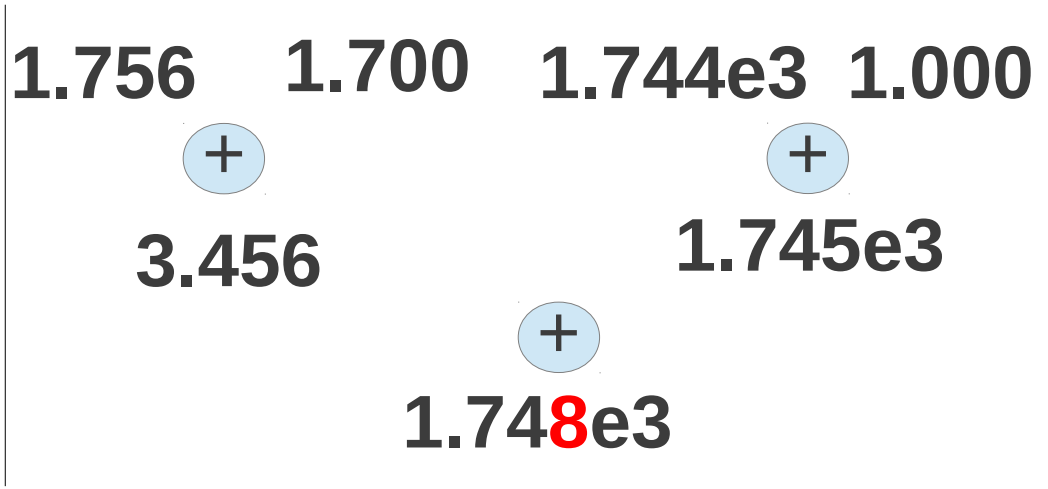


Motivation

Let's speed up arithmetic

- Floating-Point Addition:
  - Task: Want to add N numbers together
  - Solution: Build a binary tree of depth log(N)
    - Does not work!

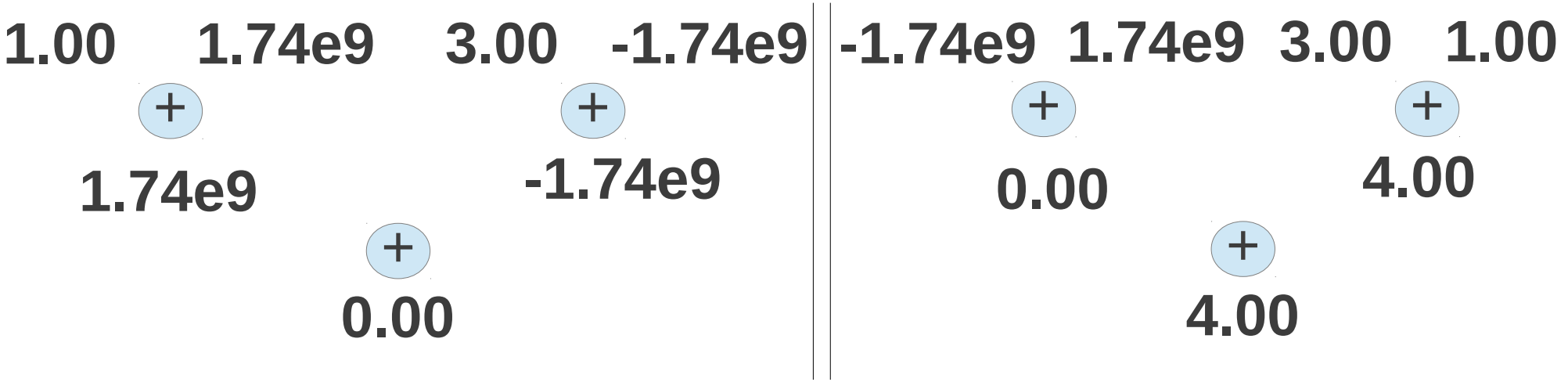




Motivation

Floating-Point Addition is not associative

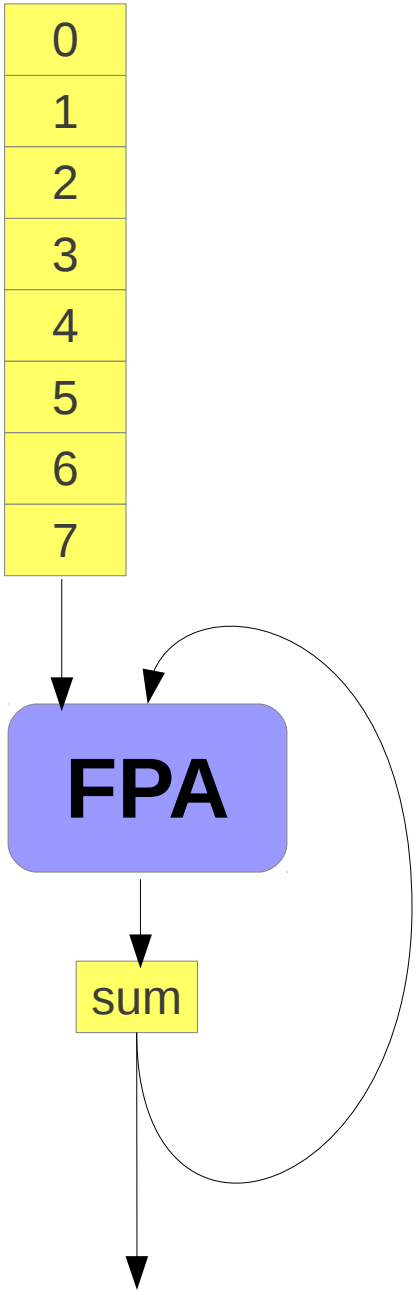
- Another example with *catastrophic cancellation*:



Infinite error!!

Motivation

- We want a ***deterministic*** answer
  - Traditionally
    - Software specifies sequential implementation
    - So implement hardware sequentially
  - O(N) depth instead of O(logN)
- Major slowdown in implementations
- Slow, no guarantee of ***accuracy***



Motivation

- In our work, we thus address all of these issues:
  - Parallelism
  - Determinism
  - Accuracy
    - Accurate rounding
- Big idea: Find an ***accurate*** answer, which implies ***determinism***, then modify operations to extract ***parallelism*** and gain speed
    - If it should be O(logN), then find a way to make it O(logN)

This Talk

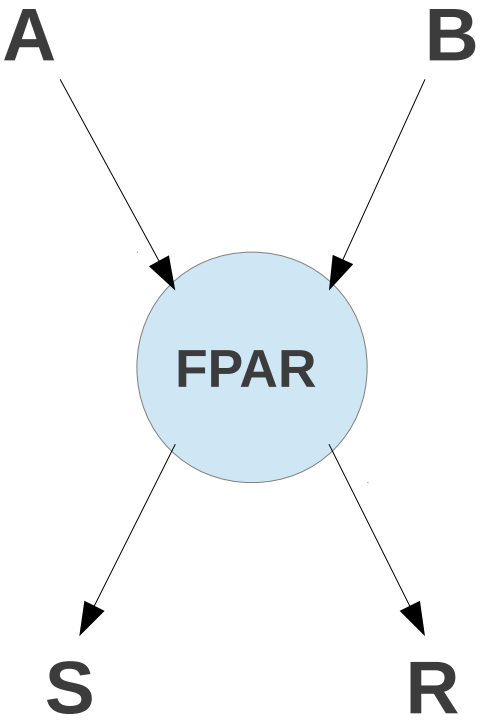
- Algorithm
  - **Accurate parallel** iterative FP addition
  - Efficient lightweight convergence detection
- Simulation Results
  - The algorithm is fast: only **2** iterations.
- Hardware Implementation
  - Double precision residue-preserving FP adder
    - Same speed as simple FPA, only **1.5x** the area
  - The overall algorithm

Outline

- Motivation
- **Algorithm**
- Simulation results
- Hardware Implementation
- Future work
- Conclusion

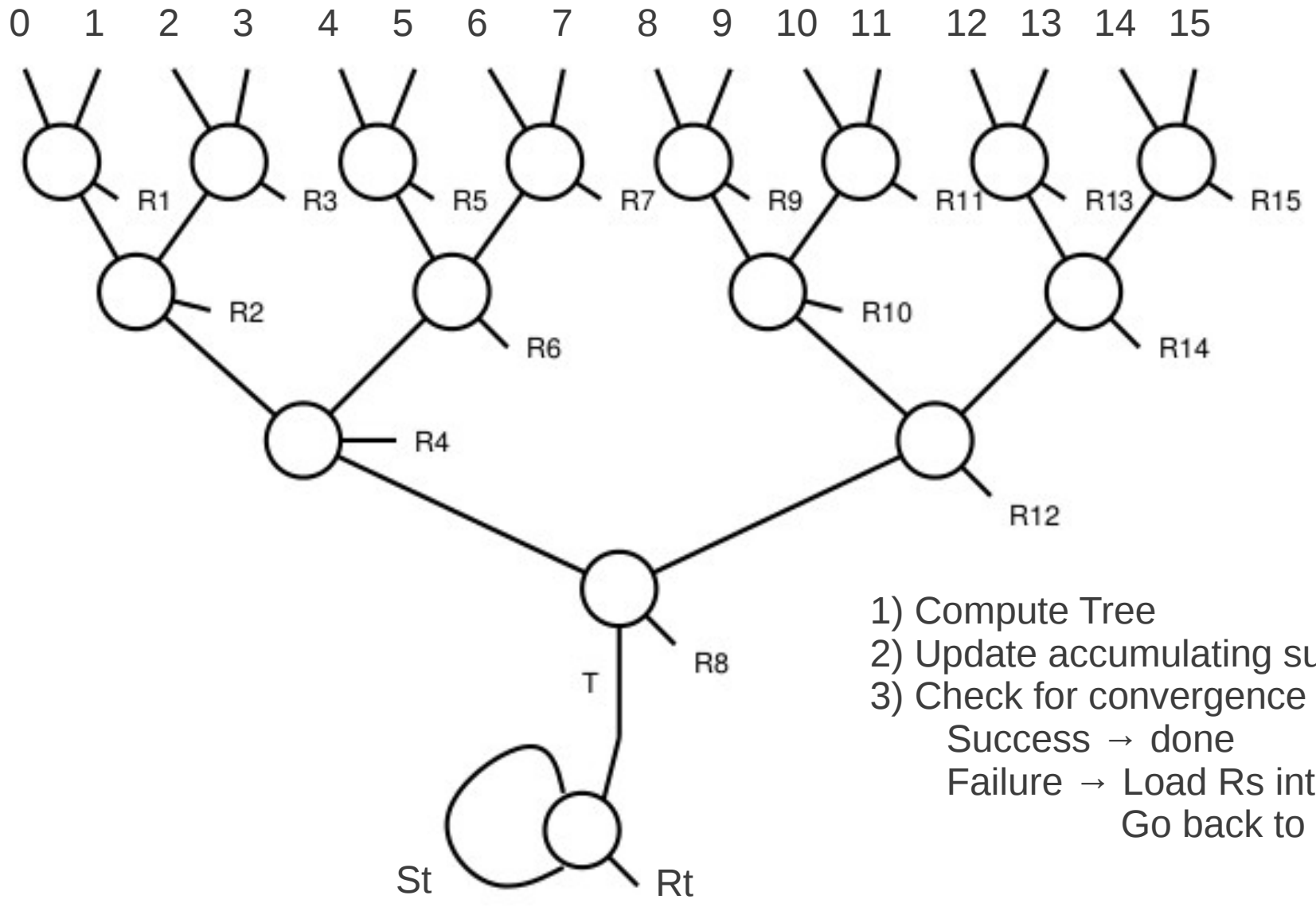
Algorithm

- Key component: ***FPAR***
  - Floating-Point Adder with Residue
    - S = Round(A+B)
    - R = (A+B)-S
- No loss of information
- Also used in prior accurate Floating-Point summation work
- Knuth provides 6 instruction software implementation



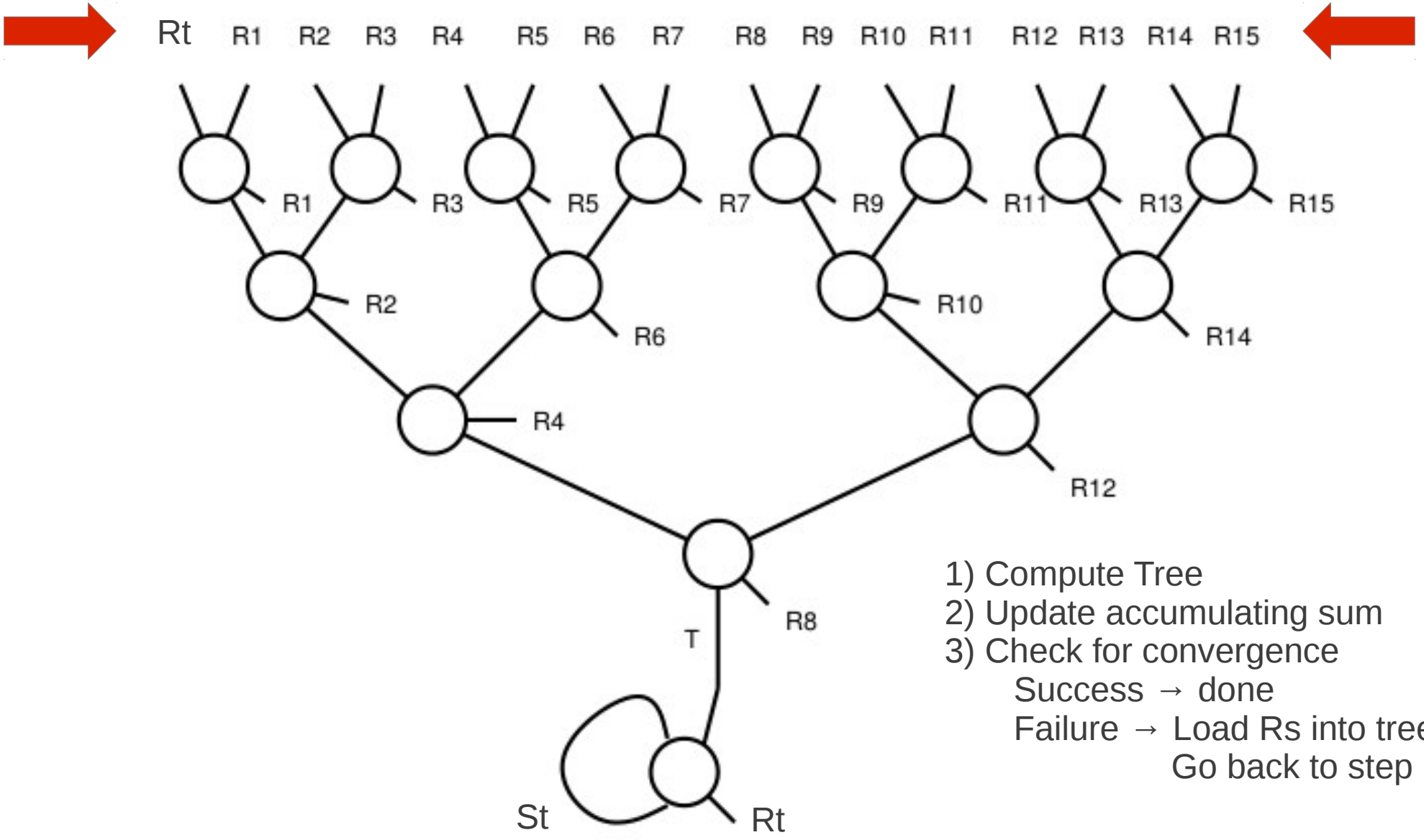
- D. E. Knuth, The Art of Computer Programming: Semi-numerical Algorithms, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997, vol. 2

Algorithm

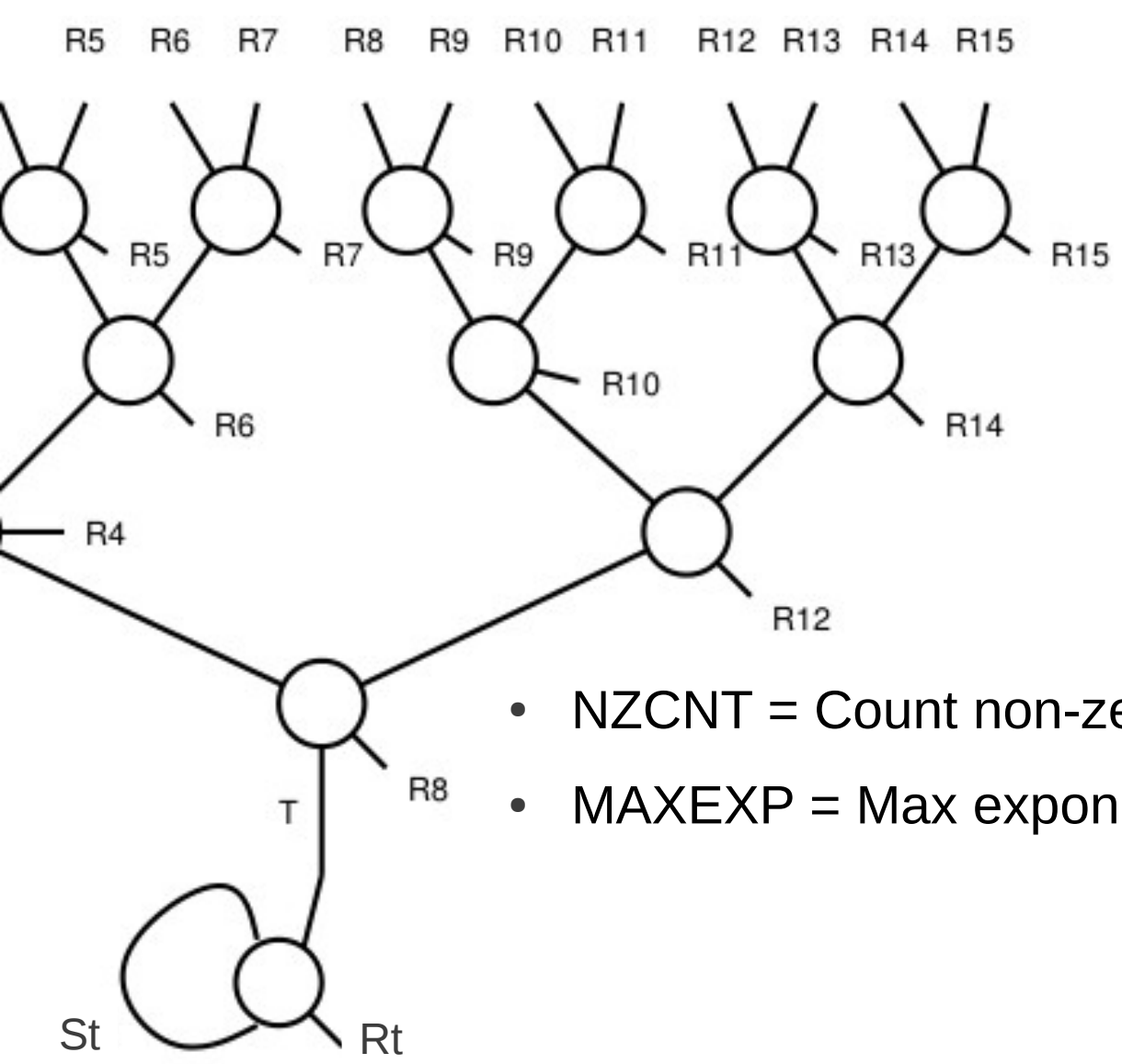


1) Compute Tree
2) Update accumulating sum
3) Check for convergence
   Success → done
   Failure → Load Rs into tree
   Go back to step 1

Algorithm



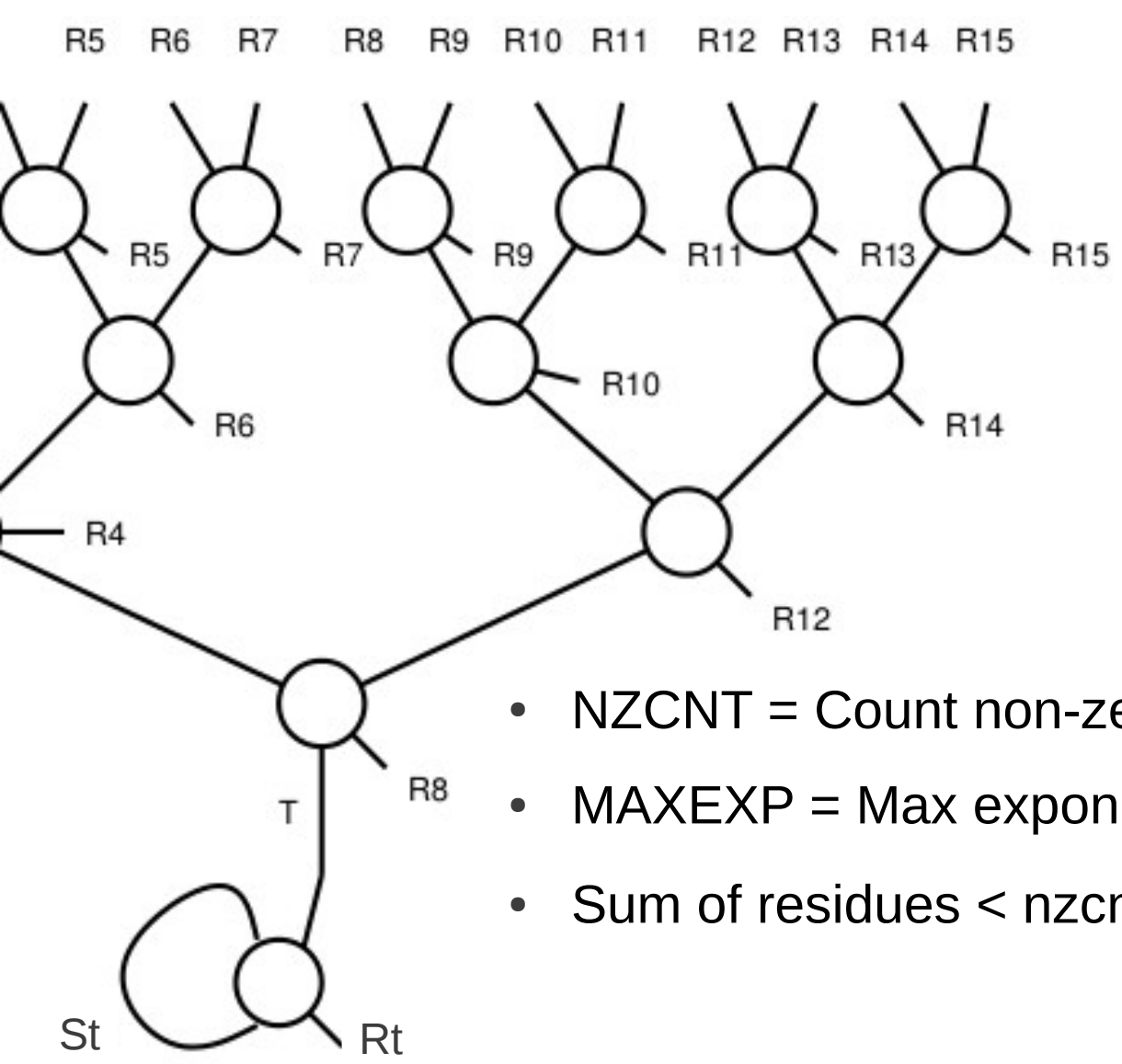
1) Compute Tree
2) Update accumulating sum
3) Check for convergence
   Success → done
   Failure → Load Rs into tree
   Go back to step 1



Algorithm

Termination detection

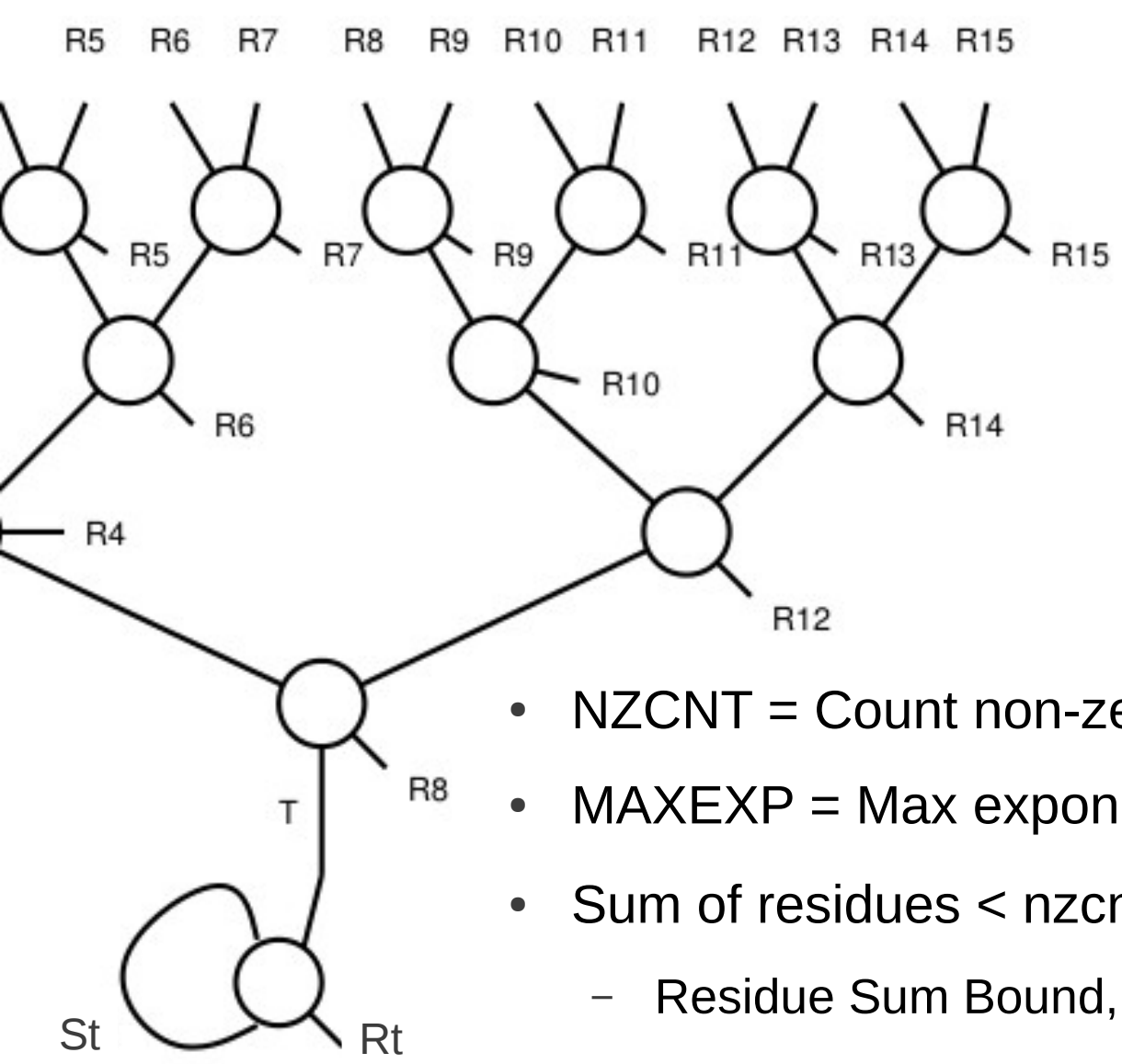
- NZCNT = Count non-zeros in R1-R15
- MAXEXP = Max exponent in R1-R15



Algorithm

Termination detection

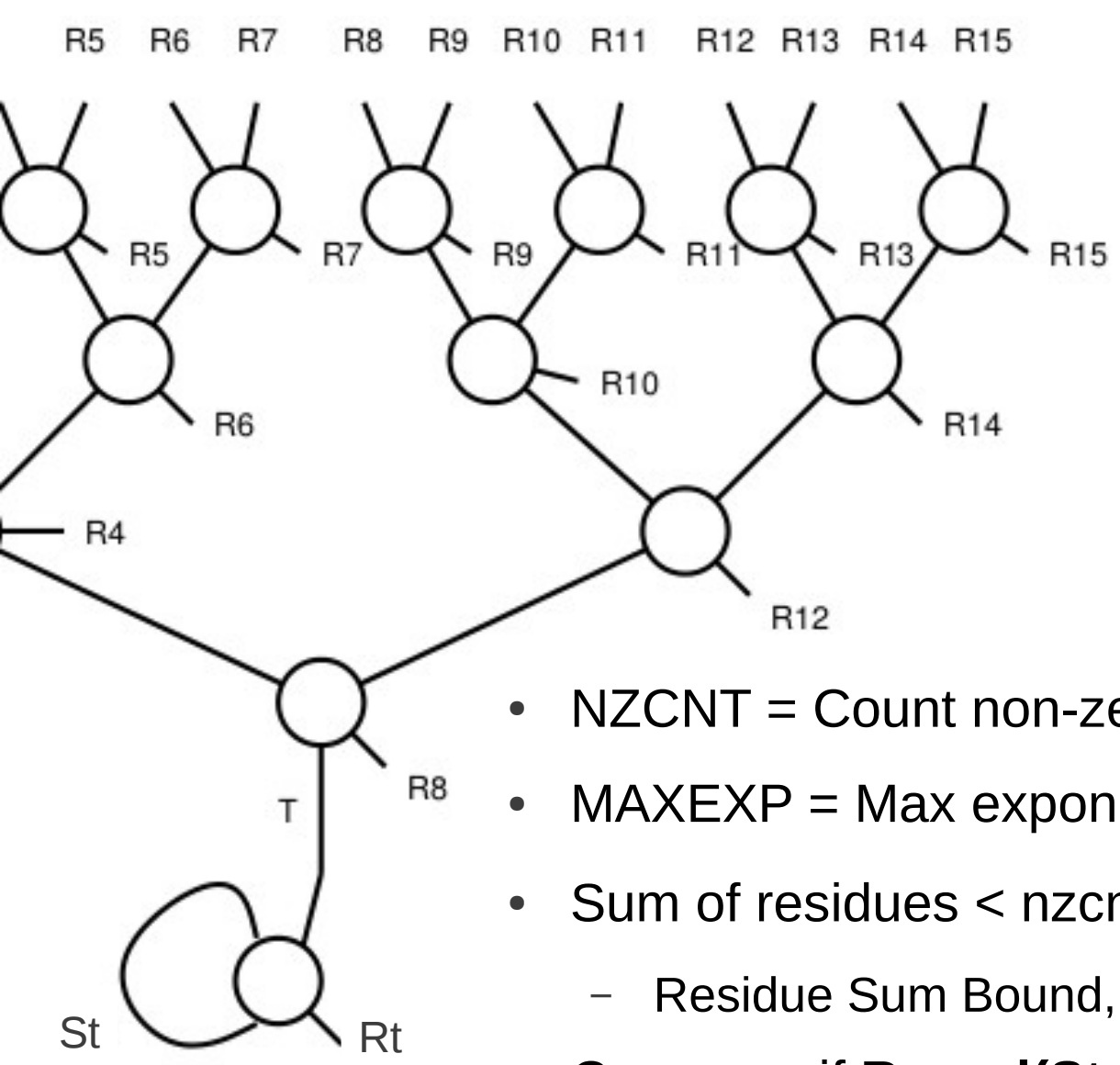
- NZCNT = Count non-zeros in R1-R15
- MAXEXP = Max exponent in R1-R15
- Sum of residues < nzcnt * maxvalue



Algorithm

Termination detection

- NZCNT = Count non-zeros in R1-R15
- MAXEXP = Max exponent in R1-R15
- Sum of residues < nzcnt * maxvalue
  - Residue Sum Bound, $rsb = 2^{maxexp + \log(nzcnt) + 1}$



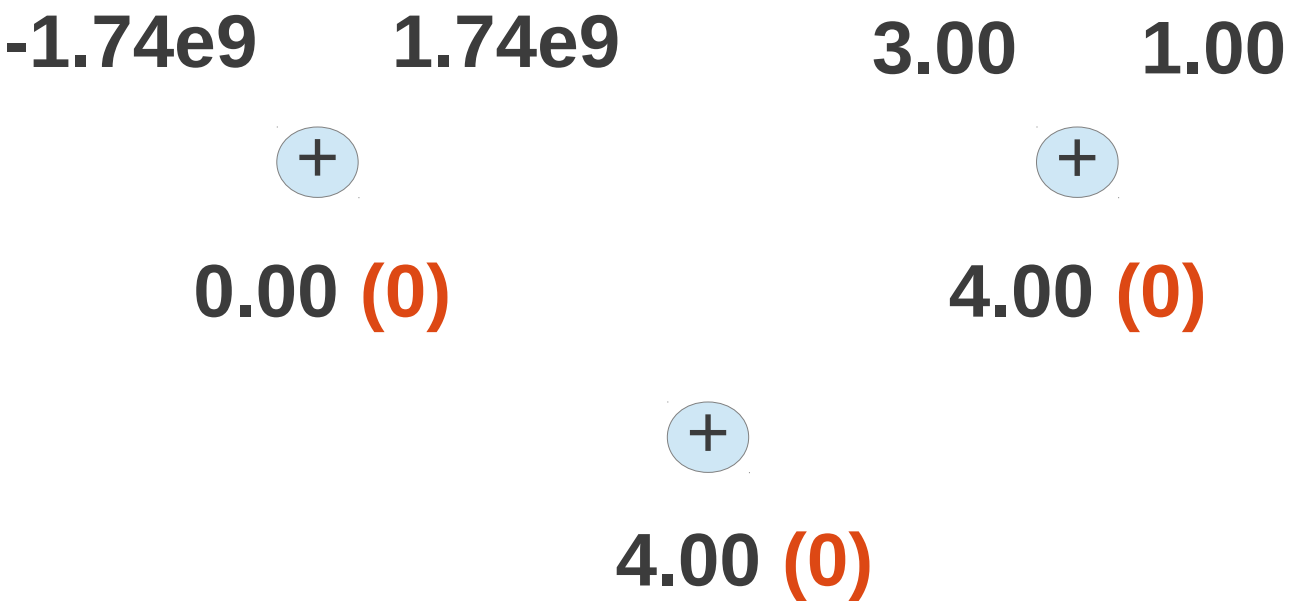
Algorithm

Termination detection

- NZCNT = Count non-zeros in R1-R15
- MAXEXP = Max exponent in R1-R15
- Sum of residues < nzcnt * maxvalue
  - Residue Sum Bound, $rsb = 2^{maxexp + \log(nzcnt) + 1}$
- Converge if **Round(St+Rt+rsb)=Round(St+Rt-rsb)**
  - Return St, the **accurate sum**
- If condition fails, then do another iteration

Going back to the previous example

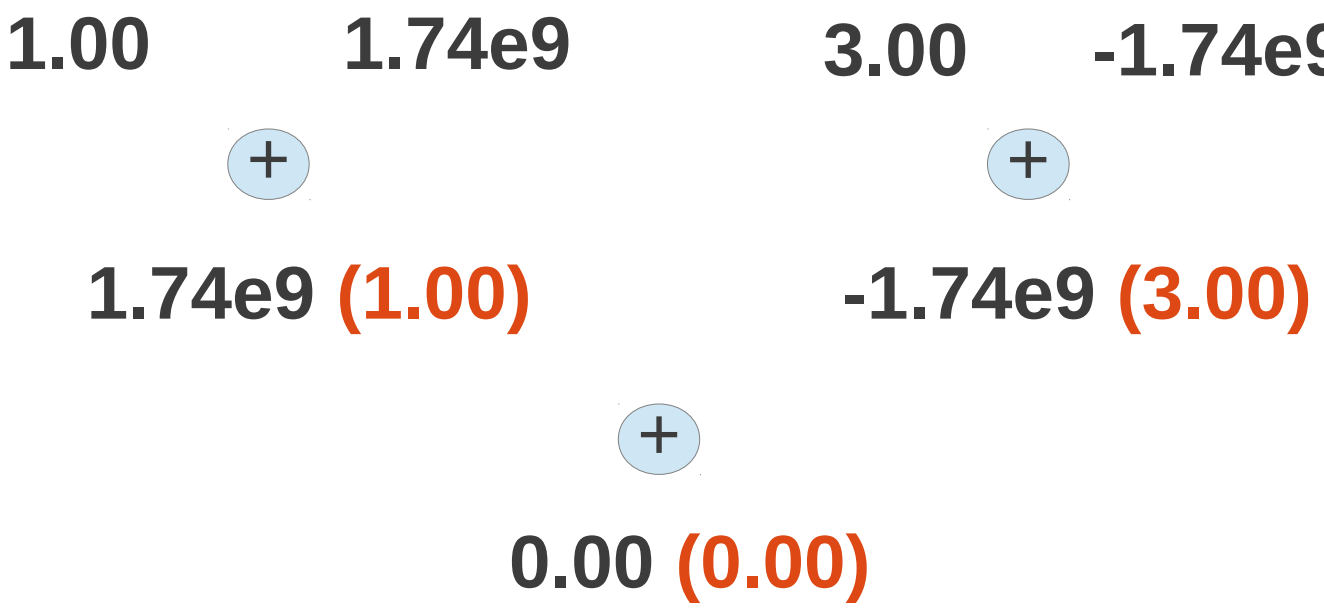
We can **make** Floating-Point Addition associative



No residues left → converge

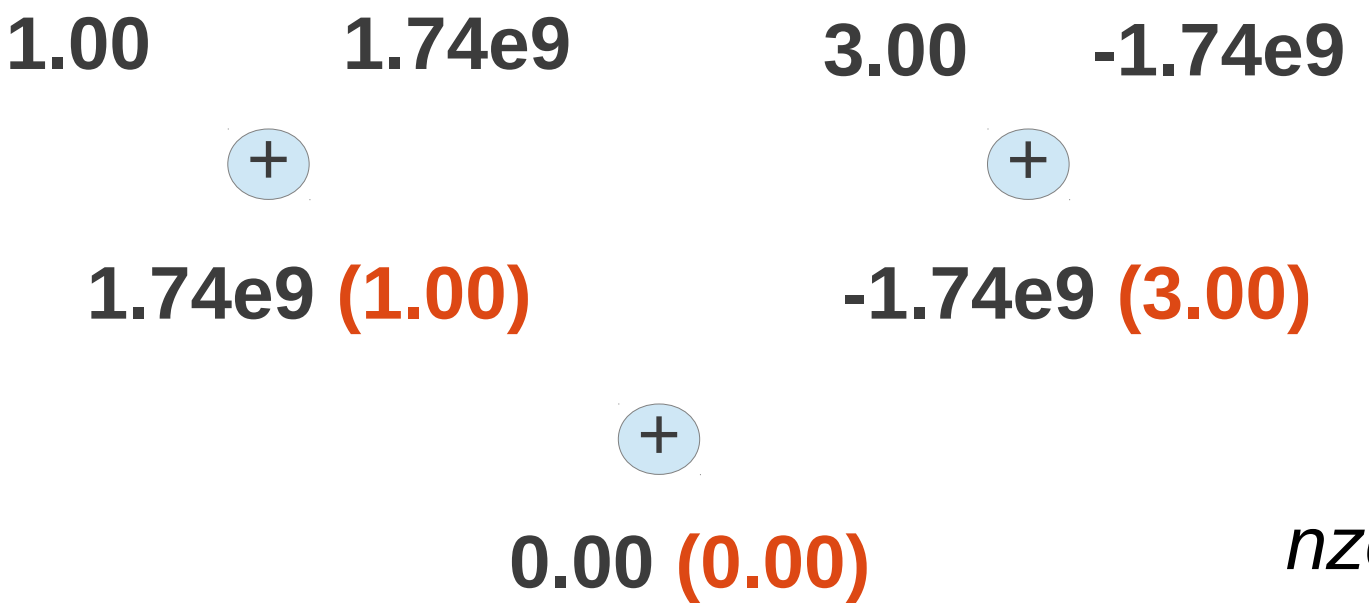
Going back to the previous example

We can **make** Floating-Point Addition associative



Going back to the previous example

We can **make** Floating-Point Addition associative



nzcnt = 2

maxexp = 0

Going back to the previous example

We can **make** Floating-Point Addition associative

1.00 **1.74e9** **3.00** **-1.74e9**

\+ \+

1.74e9 (1.00) **-1.74e9 (3.00)**

\+

0.00 (0.00)

nzcnt = 2

maxexp = 0

$$rsb = 2^{maxexp + \log(nzcnt) + 1} = 2^{0+2+1} = 8$$

Going back to the previous example

We can **make** Floating-Point Addition associative

1.00 **1.74e9** **3.00** **-1.74e9**

1.74e9 (1.00) **-1.74e9 (3.00)**

0.00 (0.00)

nzcnt = 2

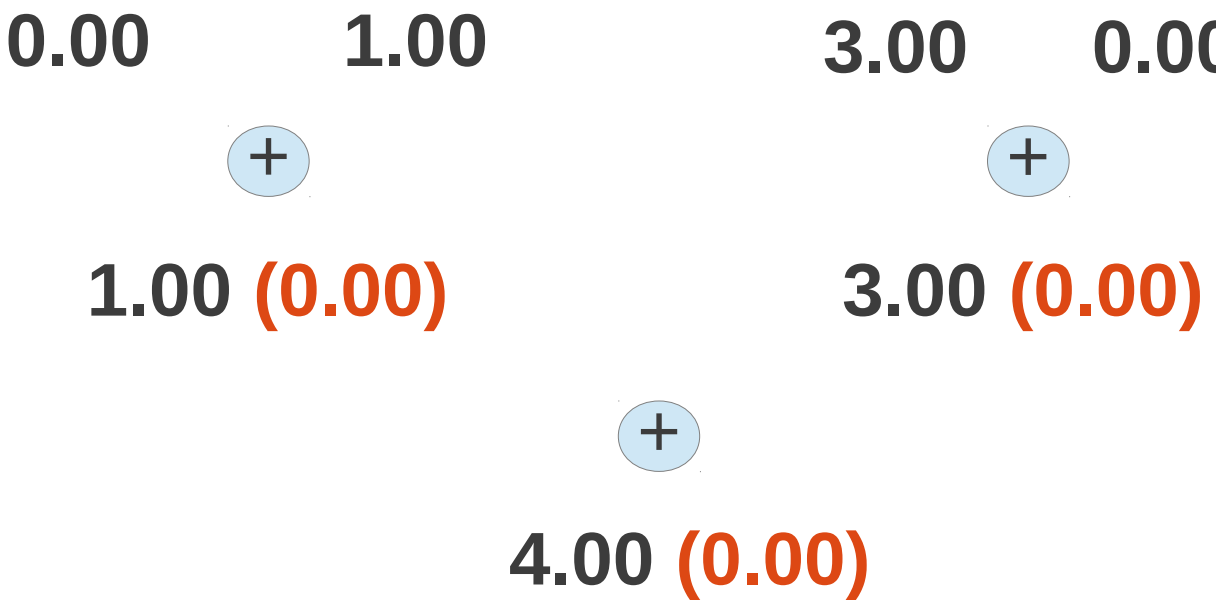
maxexp = 0

$$rsb = 2^{maxexp + \log(nzcnt) + 1} = 2^{0+2+1} = 8$$

Round(St+Rt+rsb) ≠ Round(St+Rt-rsb) → not converge

Going back to the previous example

Send residues back into the tree



After the second iteration, there are no residues left → converge

Outline

- Motivation
- Algorithm
- **Simulation results**
- Hardware Implementation
- Future work
- Conclusion

Experiments

- Iterative algorithm
  - Performance depends on input dataset
- Condition number:

$$\kappa = \frac{\sum_{i=1}^{N} |x_i|}{|\sum_{i=1}^{N} x_i|}$$

N. J. Higham, “The accuracy of floating point summation,” SIAM J. Sci. Comput, vol. 14, pp. 783–799, 1993.

- Generated 4 datasets, from κ = 1 to κ → infinity

Experiments

- Uniform distribution
  - Most numbers have high exponents
- Exponential distribution
  - Numbers across all ranges of exponents
- $N = 2^{12} = 4096$

Simulation results

Exponential distribution

Data 1 - κ = 1		
# its σ	nzcnt	s/w seq % error
2, 0	97%	3e-14

Uniform distribution

Data 1 - κ = 1		
# its, σ	nzcnt	s/w seq % error
2, 0	0%	1e-13

Simulation results

Number of tree iterations: **2**, in virtually all cases

Exponential distribution

Data 1 - κ = 1			Data 2 - κ = 78 on avg			Data 3 - κ = 6e16 on avg			Data 4 - κ → ∞		
# its σ	nzcnt	s/w seq % error	# its, σ	nzcnt	s/w seq % error	# its, σ	nzcnt	s/w seq % error	# its, σ	nzcnt	s/w seq % error
2, 0	97%	3e-14	2, 0	97%	1e-13	2, 0	0%	9e4	39, 0.05	0%	∞

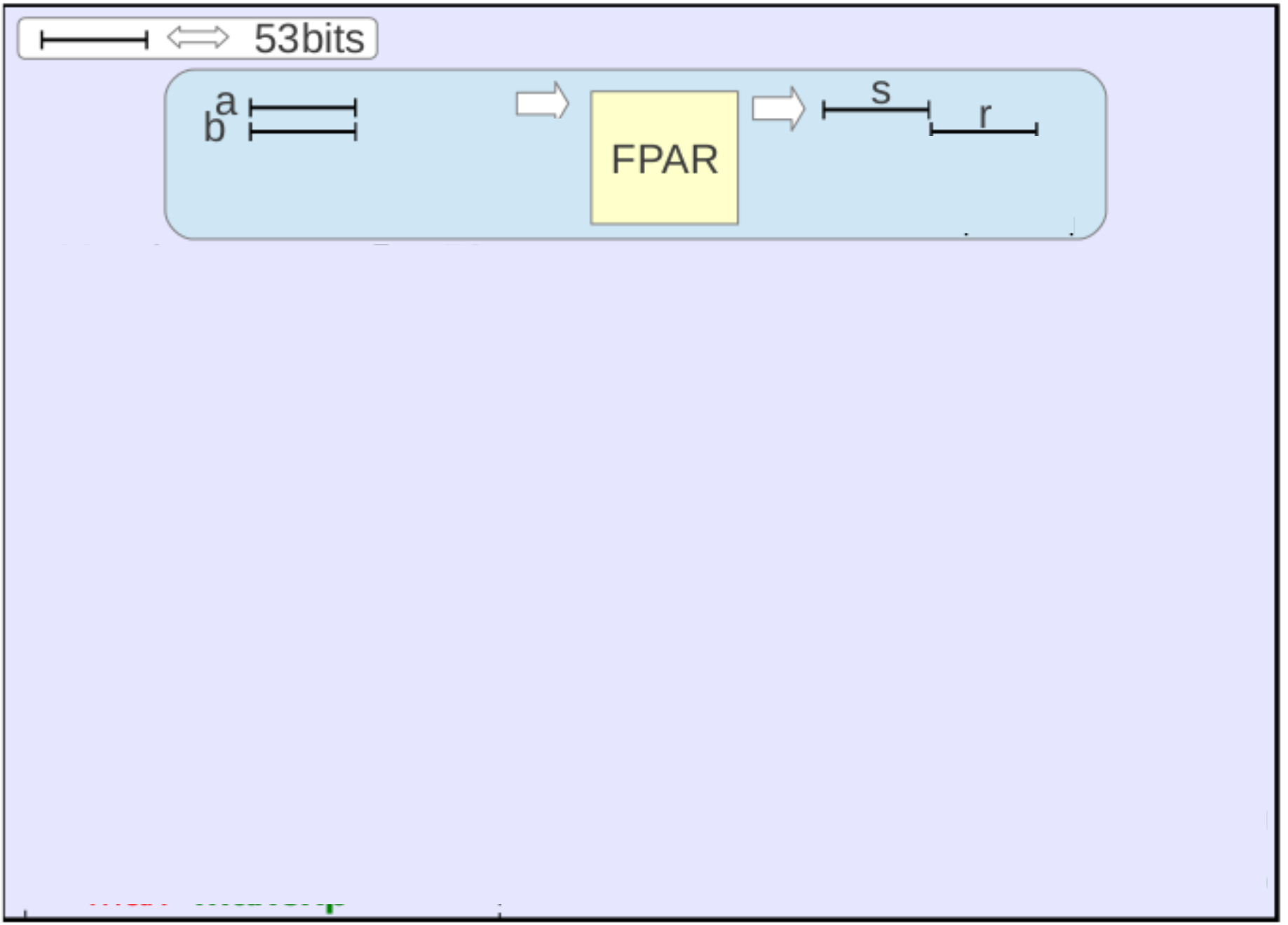
Uniform distribution

Data 1 - κ = 1			Data 2 - κ = 330 on avg			Data 3 - κ = 2e17 on avg			Data 4 - κ → ∞		
# its, σ	nzcnt	s/w seq % error	# its, σ	nzcnt	s/w seq % error	# its, σ	nzcnt	s/w seq % error	# its, σ	nzcnt	s/w seq % error
2, 0	0%	1e-13	2, 0	0%	8e-13	2, 0	0%	2	2, 0	0%	∞

$$\kappa = \frac{\sum_{i=1}^{N} |x_i|}{|\sum_{i=1}^{N} x_i|}$$

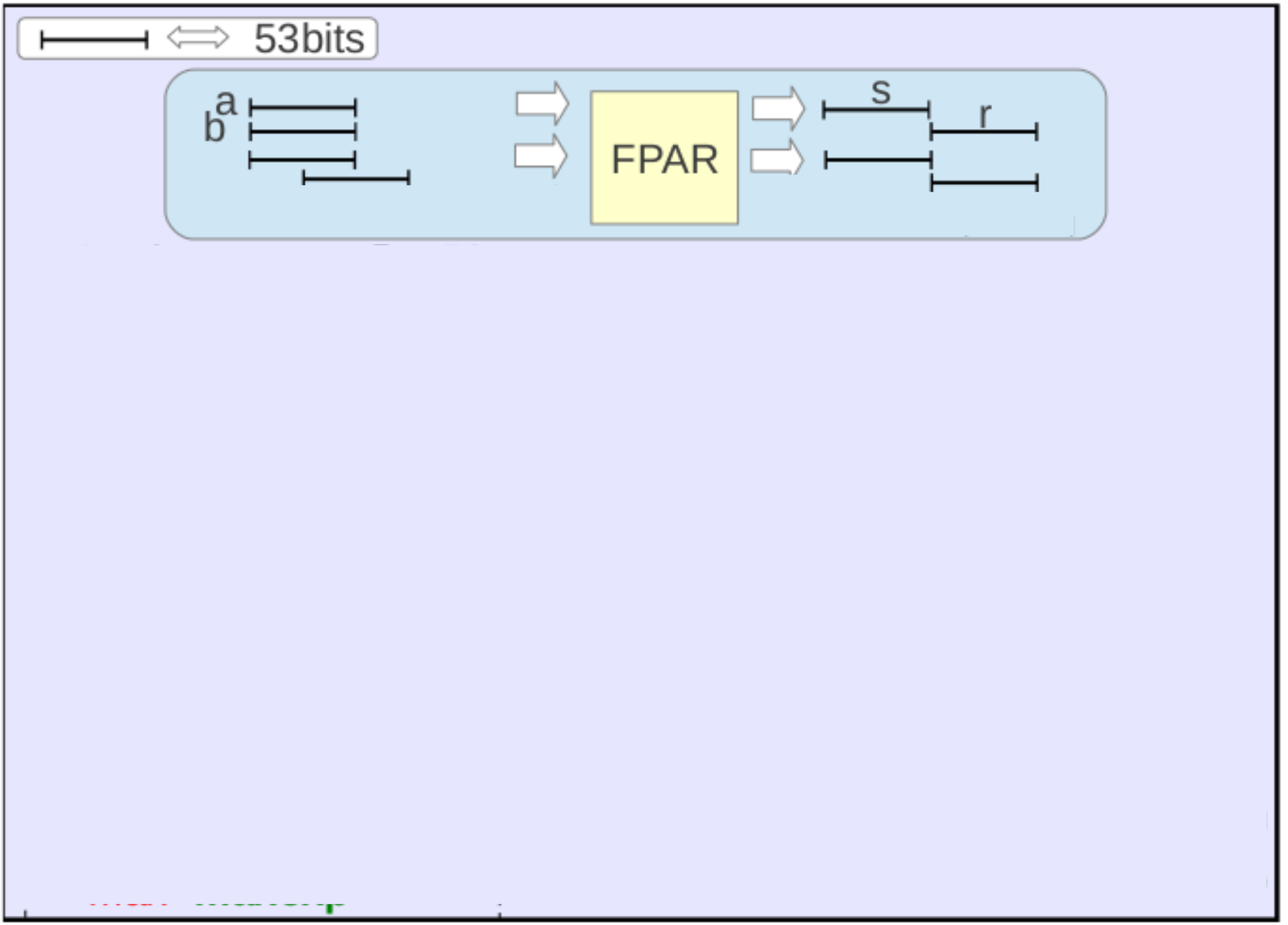
Simulation results

Why **2** iterations?



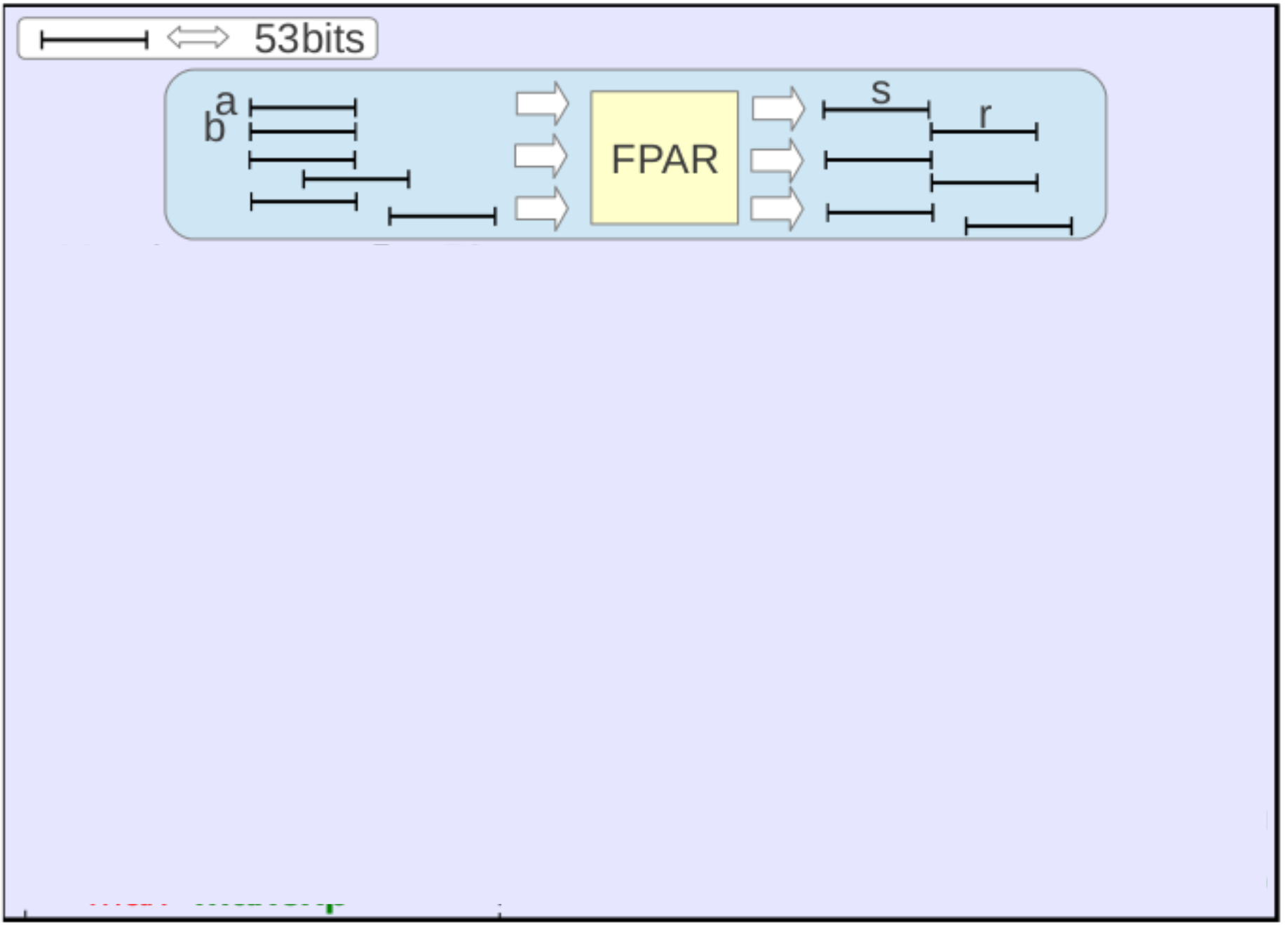
Simulation results

Why **2** iterations?



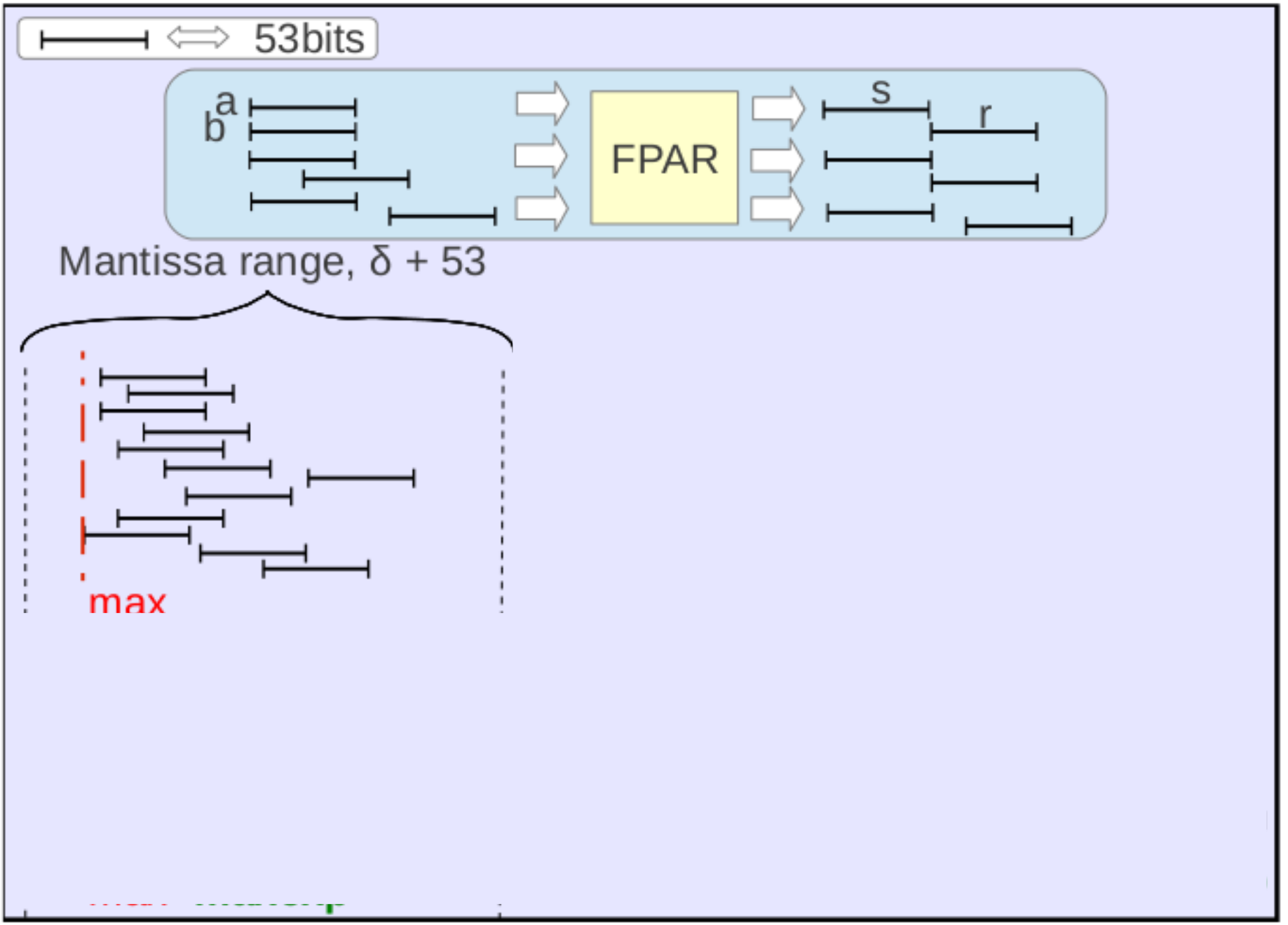
Simulation results

Why **2** iterations?



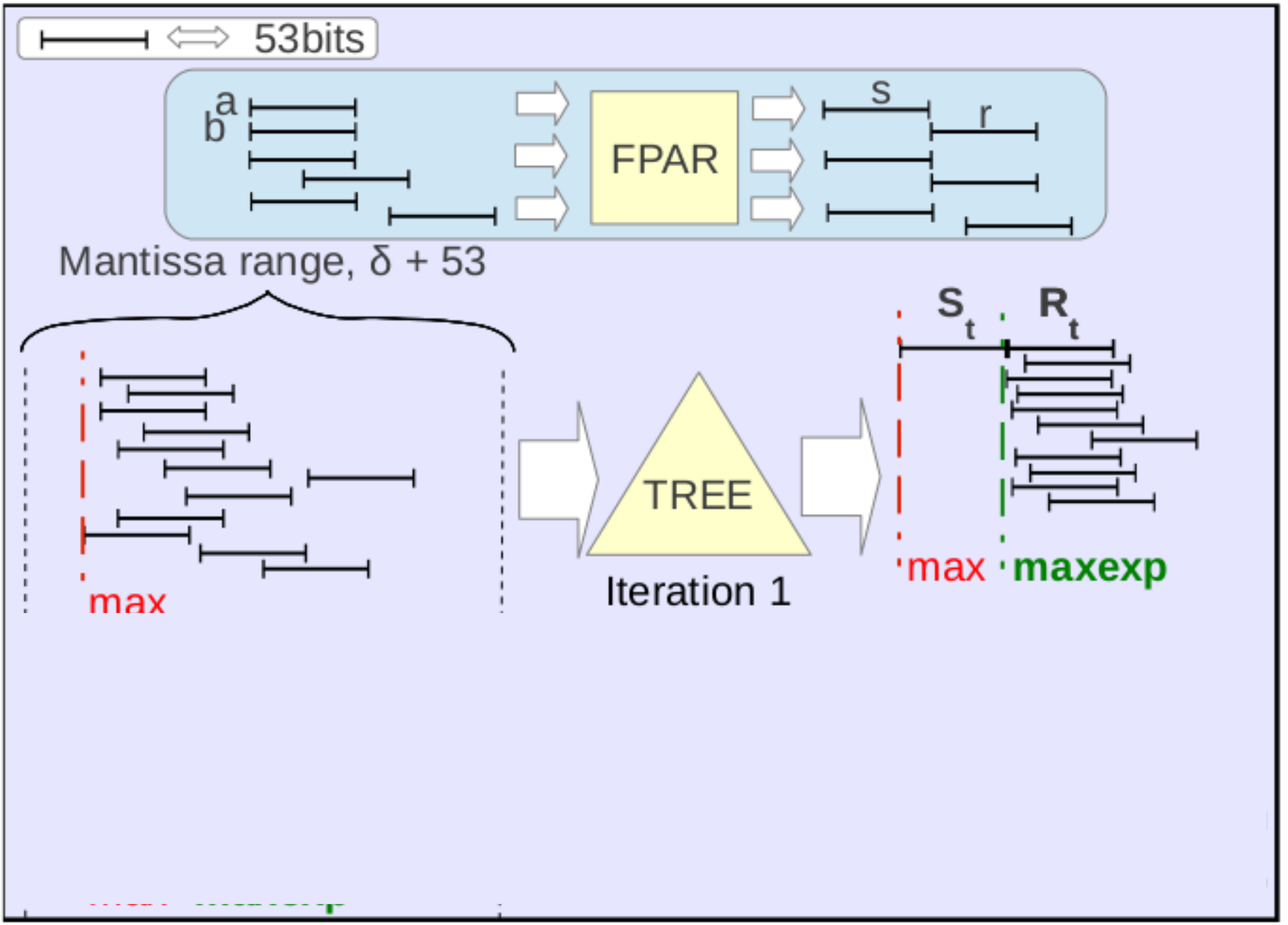
Simulation results

Why **2** iterations?



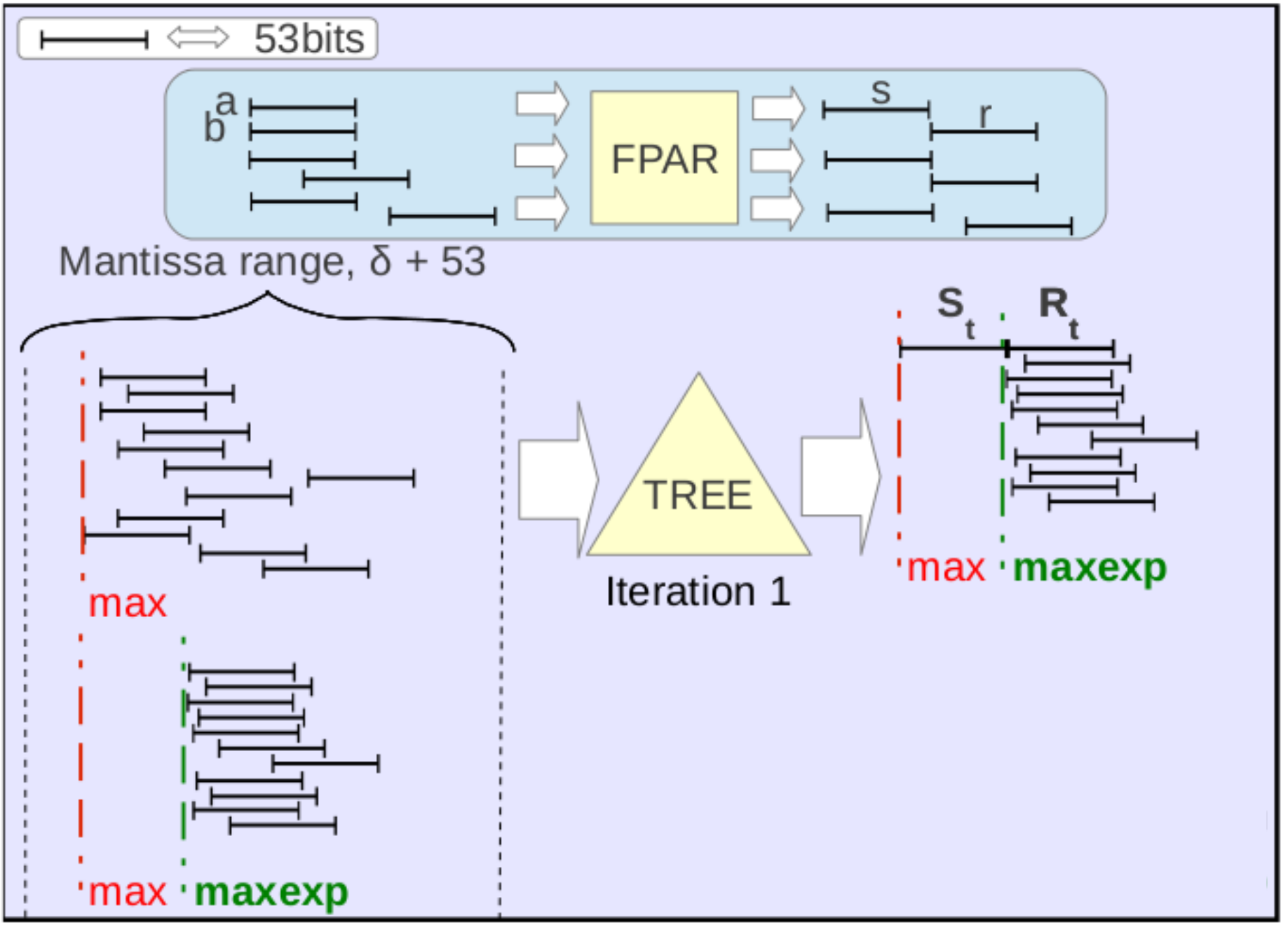
Simulation results

Why 2 iterations?



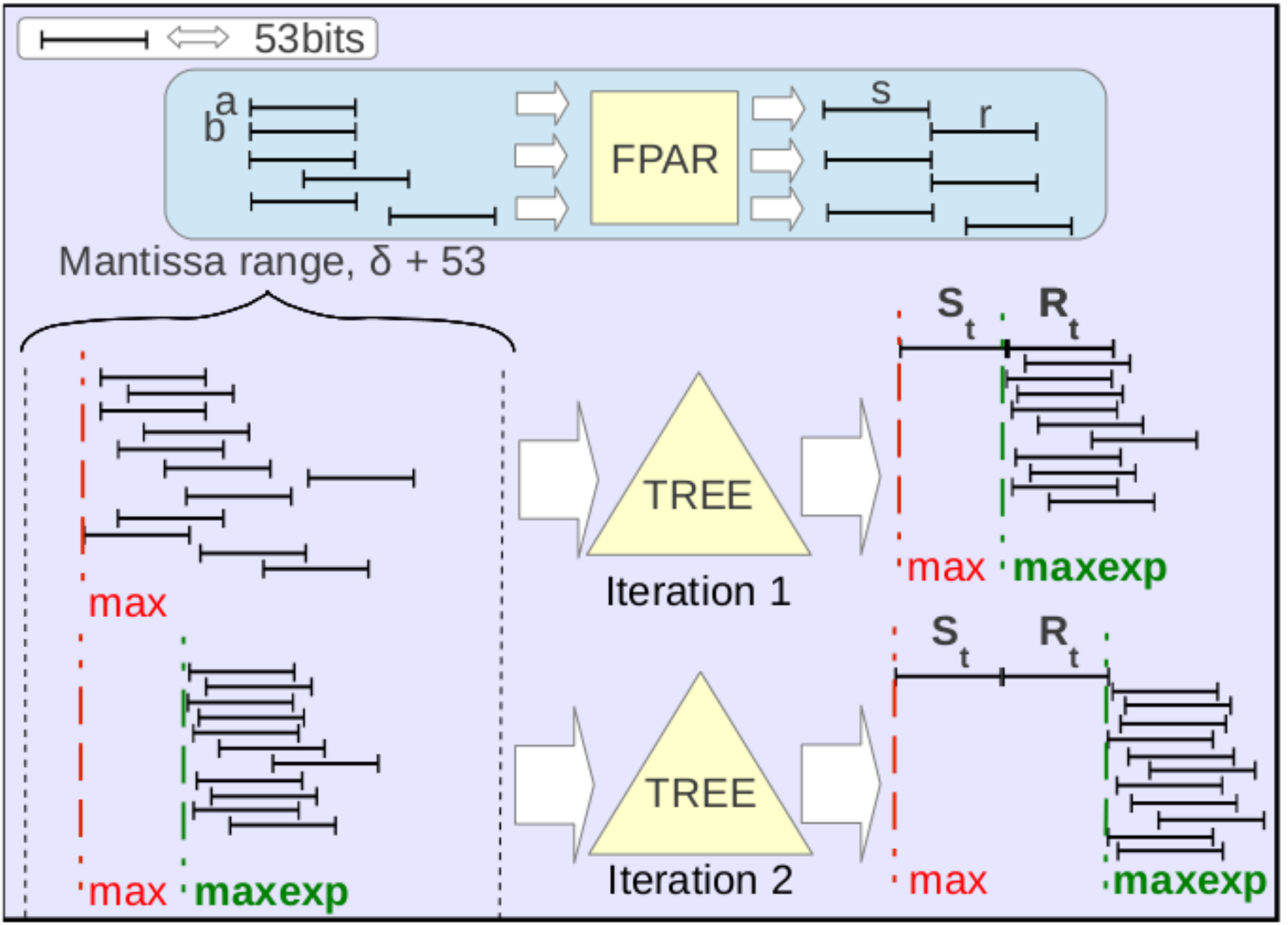
Simulation results

Why 2 iterations?



Simulation results

Why 2 iterations?

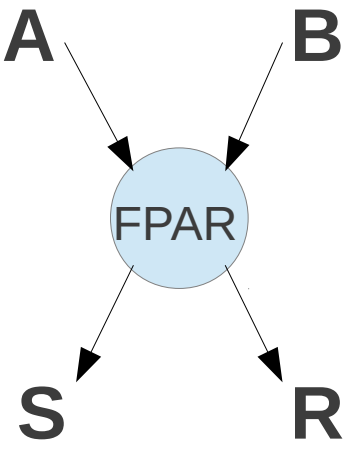


Accurate Parallel Floating-Point Accumulation --- *Arith21*, April 2013

Outline

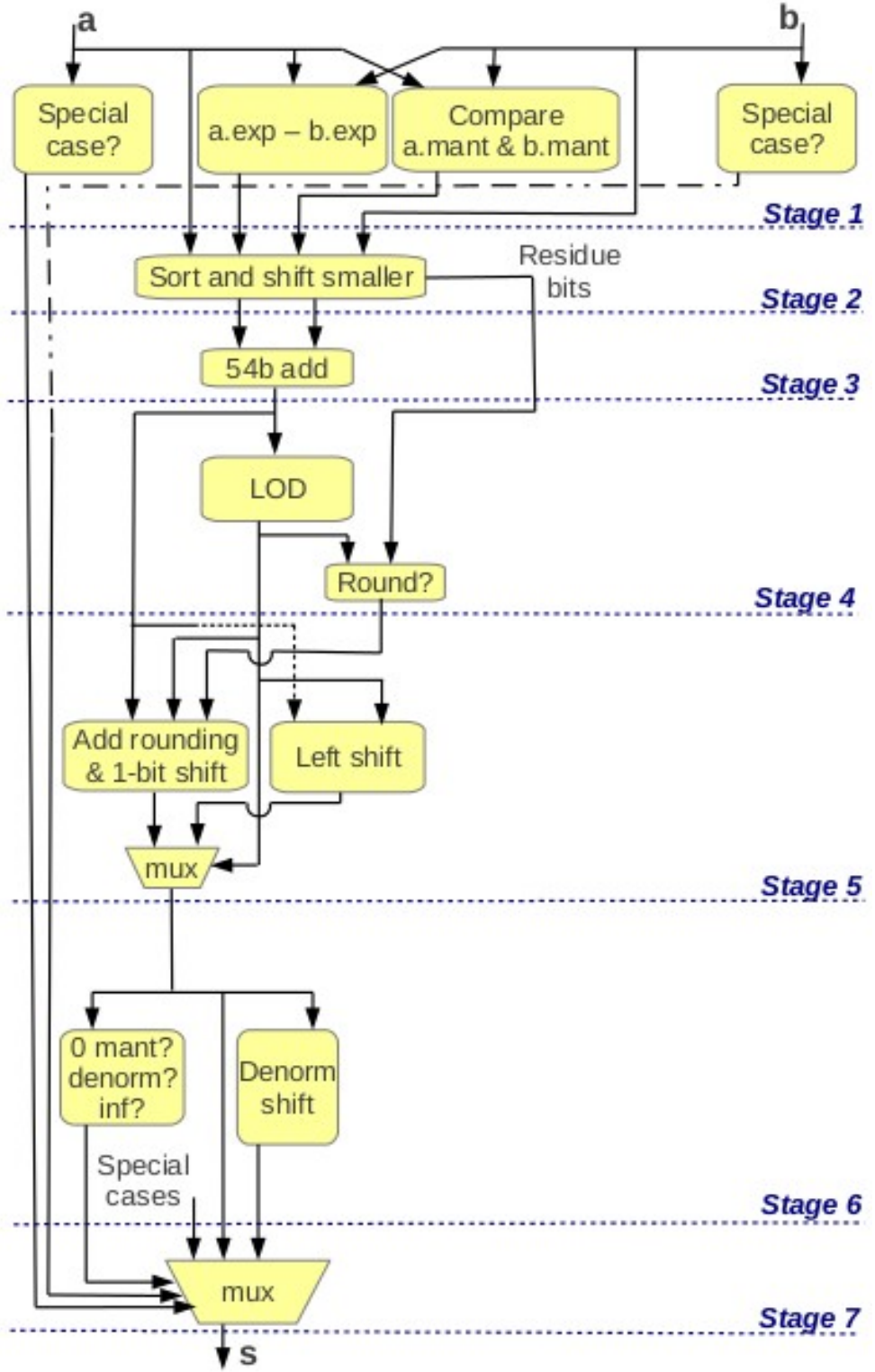
- Motivation
- Algorithm
- Simulation results
- **Hardware Implementation**
- Future work
- Conclusion

FPAR Implementation



- Remember our building block, the ***FPAR***
- Previous accurate accumulation algorithms use it too
  - Software implementation
  - Uses FP adder in commodity FPU
  - Requires 6 instructions, depth of 5
- We provide a double-precision implementation

$$(s, r) = Knuth_{fpar}(a, b):$$
$$s = a + b$$
$$b' = s - a$$
$$a' = s - b';\ \delta_b = b - b'$$
$$\delta_a = a - a'$$
$$r = \delta_a + \delta_b$$



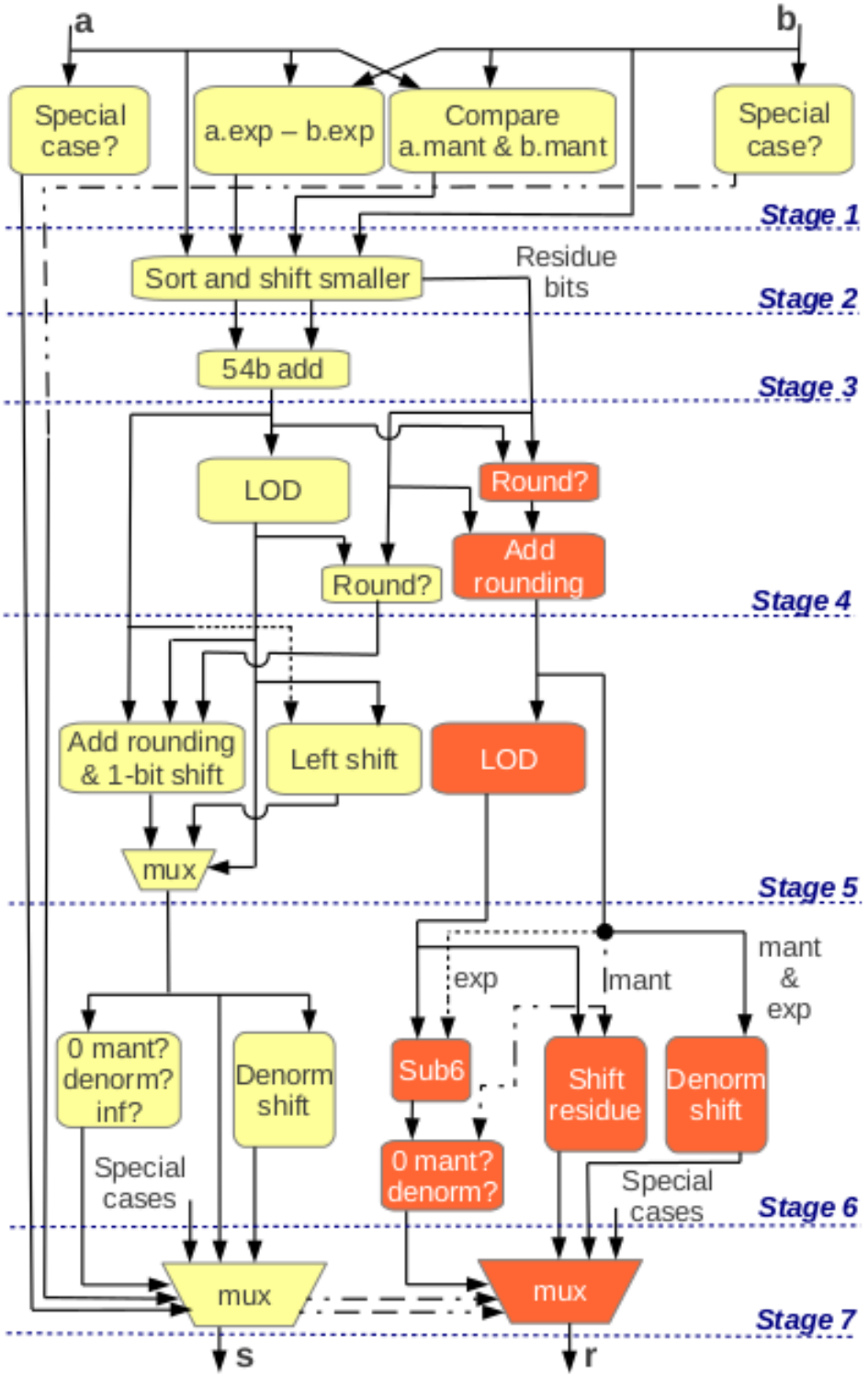
FP adder Implementation

FPA

Floating-Point Adder

- On ML605, Virtex 6
- 250MHz
- 7 pipeline stages
- 1517 LUTs

- Similar to improved single-path floating-point adder in:
- M. Ercegovac and T. Lang, Digital Arithmetic, ser. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, 2003.



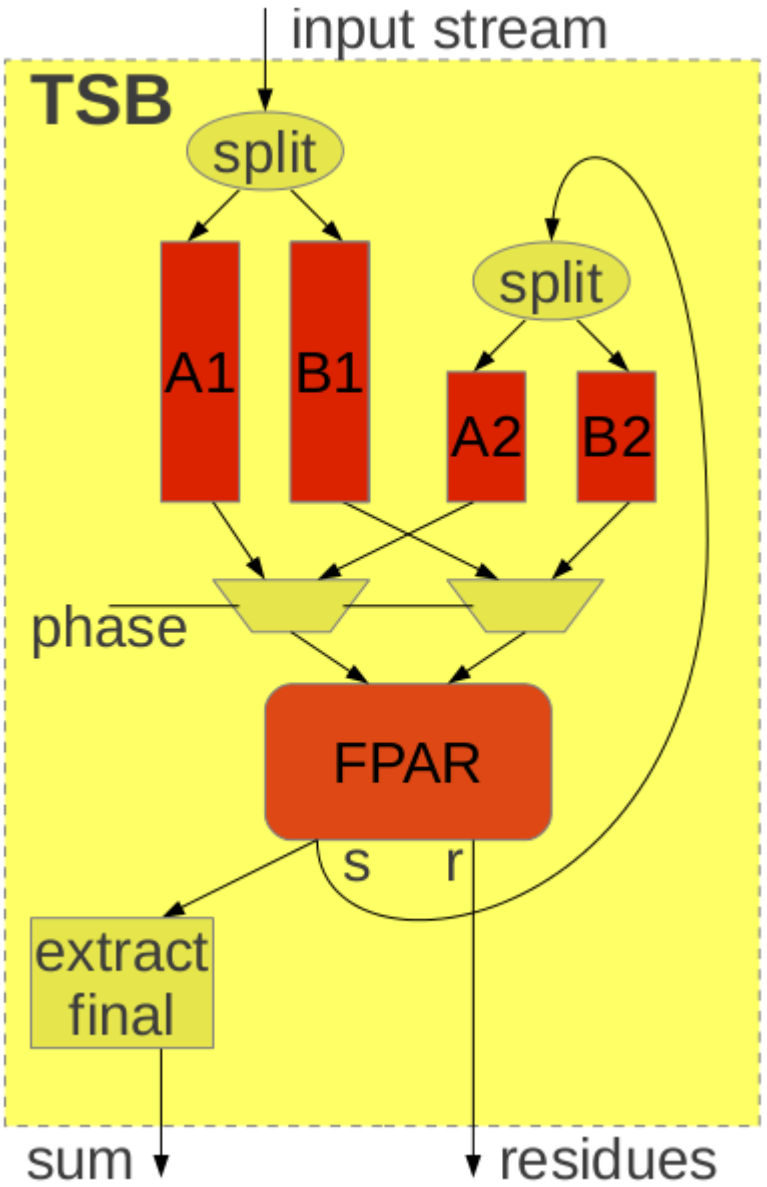
FPAR Implementation

FPAR

Floating-Point Adder with Residue

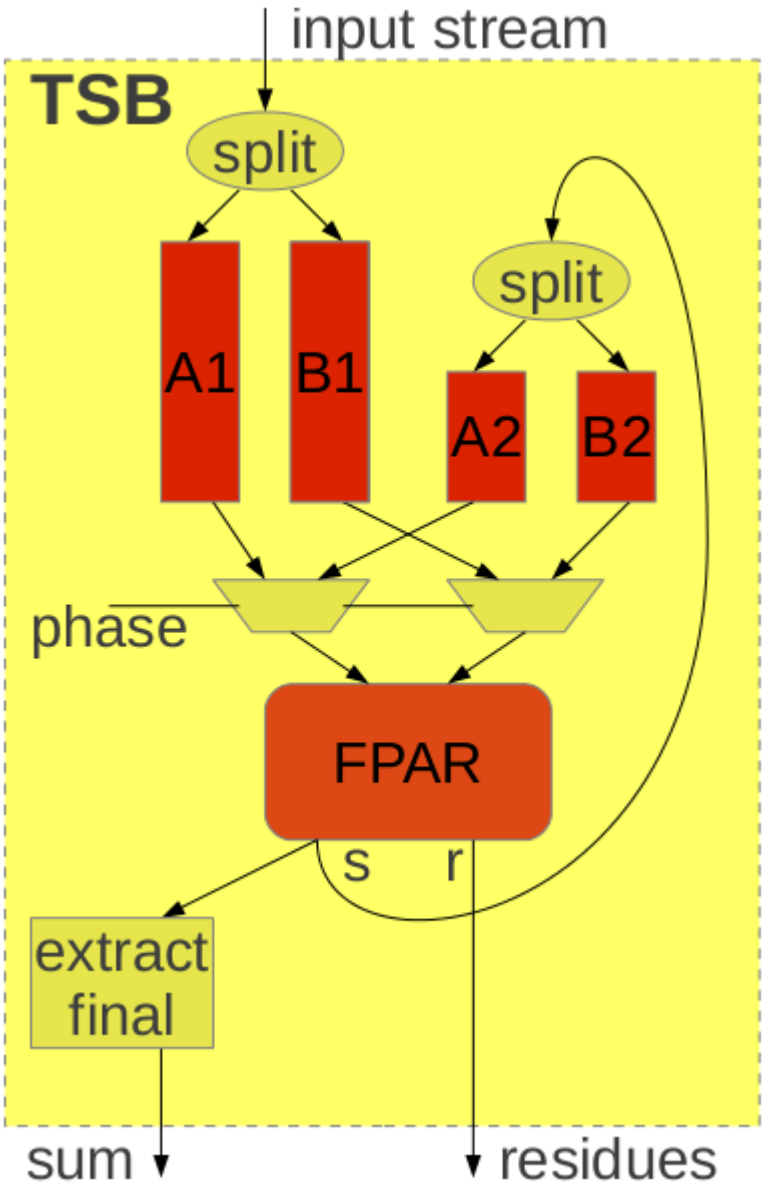
- On ML605, Virtex 6
- 250MHz
- 7 pipeline stages
- 2252 LUTs
- **Same speed**
- **Only 48% more area**

Tree Implementation



Last question: How can the tree be efficiently implemented in hardware?

Tree Implementation



Last question: How can the tree be efficiently implemented in hardware?

- Can add more FPARs to speed up computation
  - Consume m summands/cycle
  - 3N/m latency
  - O(N) work
- ***Area/Speed trade-off***

Source code

- The hardware was implemented in Bluespec
- Distribution available on our website

http://ic.ese.upenn.edu/Abstracts/parallel_fpaccum_arith2013.html

Outline

- Motivation
- Algorithm
- Simulation results
- Hardware Implementation
- **Future work**
- Conclusion

Future work

- Earlier convergence check
  - Maybe converge in less than 2 iterations?
- No rounding in FPAR
  - Less area consumption
  - Avoid extra work

Outline

- Motivation
- Algorithm
- Simulation results
- Hardware Implementation
- Future work
- **Conclusion**

Conclusion

- Common implementations of FP addition
  - Non-accurate, slow, sequential, O(N)
- Our implementation of FP addition
  - Accurate, fast, Parallel, O(logN)
  - Proved that it works (see paper)
  - Fast in practice: only 2 iterations
  - Low area overhead: <50% for FPAR unit
  - Allows us to use the increasing number of transistors on chips to increase parallelism and speed up computations

Algorithm – Big Ties

- When convergence fails, perform another iteration
- But before that, we need to check for a ***Big Tie***

St = 1.1010e27

Rt = 1.0000e22

rsb = 1.0000e10

Sum

11010 **10000**00.....±1

Round(St+Rt+rsb) = 1.1011e27

Round(St+Rt-rsb) = 1.1010e27

- Extremely rare, details in paper

Comparisons

- Comparisons

Accumulation Algorithm	FLOPs/summand	Depth
iFastSum [7] with our FPAR	12 3	$O(N)$
Simple, inaccurate	1	$O(\log(N))$
Optimistic Sequential [13]	40	$O(\log(N))$
Leuprecht tree [4] with our FPAR	14 5	$O(\log(N))$
This Work	3	$O(\log(N))$

3N + O(1)