Low Latency Digit-Recurrence Reciprocal and Square-Root Reciprocal Algorithm and Architecture

Elisardo Antelo[(1)], Tomás Lang[(2)],
Paolo Montuschi[(3)], Alberto Nannarelli[(4)]

[(1)] Dept. Electronic & Computer Eng., University of Santiago de Compostela. Spain
[(2)] Dept. Electrical & Computer Eng., University of California, Irvine. USA
[(3)] Dept. Automatica e Informatica, Politecnico di Torino. Italy
[(4)] Dept. Informatics & Math. Modelling, Technical University of Denmark. Denmark

Outline

- Motivation and Background
- Reciprocal Algorithm and Implementation
- Square-Root Reciprocal Algorithm and Implementation
- Evaluation and Comparisons
- Conclusions and Future Directions

Use of $\frac{1}{d}$ and $\frac{1}{\sqrt{d}}$

$\frac{1}{d}$ and $\frac{1}{\sqrt{d}}$ are common operations in

- Computer Graphics (geometry engine)

  Examples:
  - Sony's Emotion Engine (PS2)
  - Cell processor

- Scientific computations

  Vector operations:

  vector division, vector normalization, . . .

Hardware Impl. of $\frac{1}{d}$ and $\frac{1}{\sqrt{d}}$

Alternatives for hardware implementation of $\frac{1}{d}$ and $\frac{1}{\sqrt{d}}$

Linear convergence

- Digit–by–digit algorithms

ADV. low hardware overhead

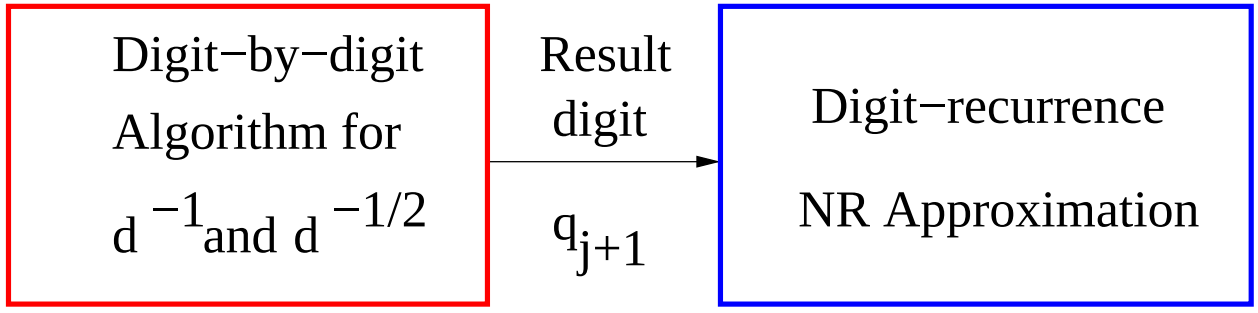
Quadratic convergence

- Quadratic convergent (Newton-Raphson) algorithm
- Polynomial approximations

ADV. reduced number of iterations
DIS. dedicated parallel multiplier and tables required

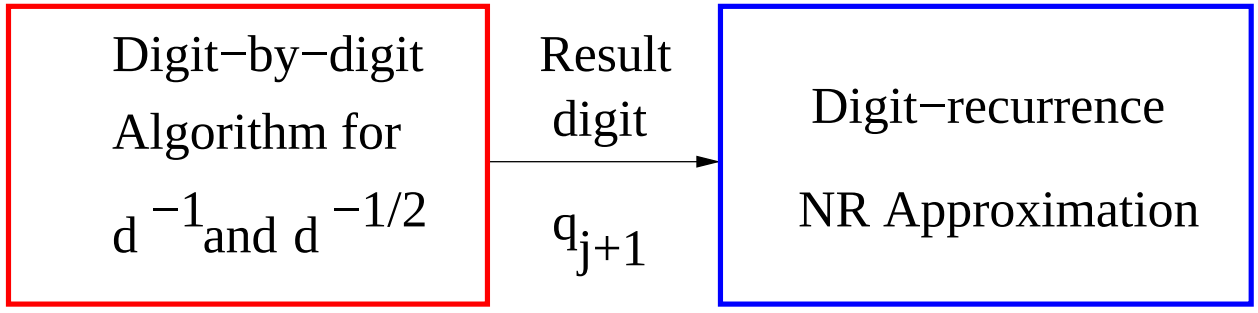
Proposed Algorithm

- Digit–by–digit algorithm together with one iteration NR

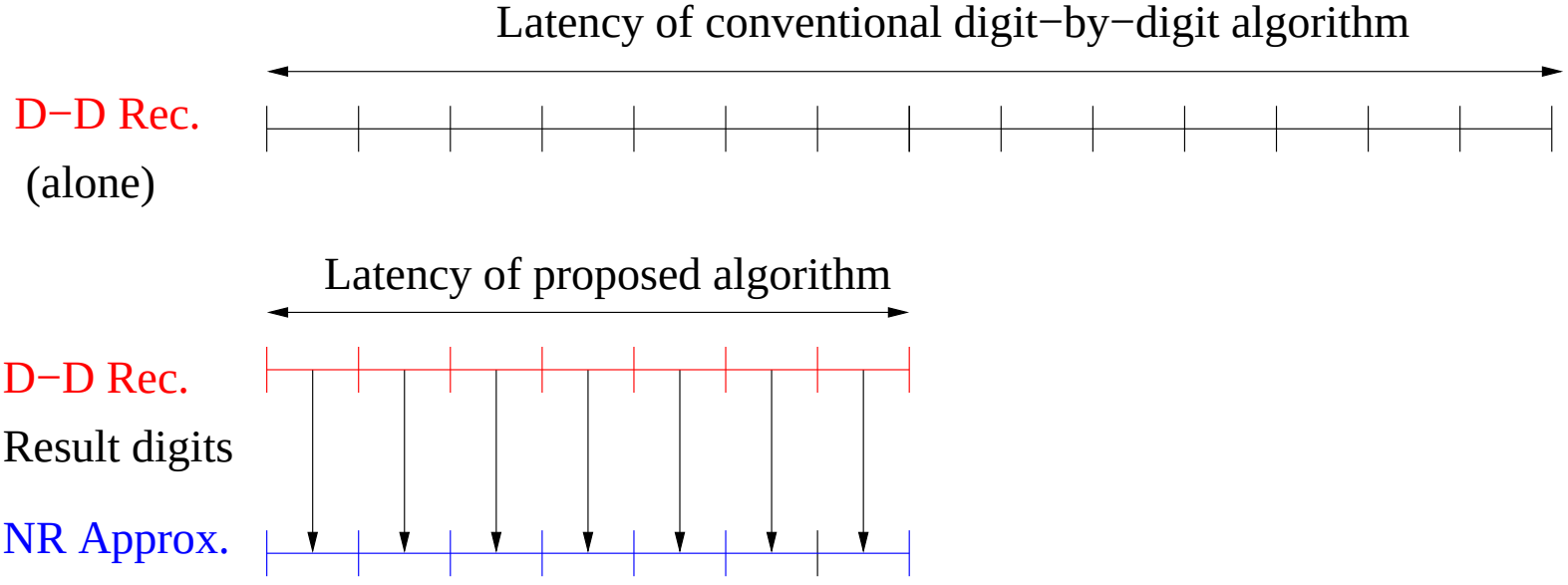


Proposed Algorithm

- Digit–by–digit algorithm together with one iteration NR



- Overlapped execution



One Newton-Raphson Iteration

$\frac{1}{d}$: k approximation of $\frac{1}{d}$

$$A = k(2 - kd)$$

error on k: $\delta = 1 - kd$
error on A: $\epsilon = 1 - Ad = \delta^2$

$\frac{1}{\sqrt{d}}$: k approximation of $\frac{1}{\sqrt{d}}$

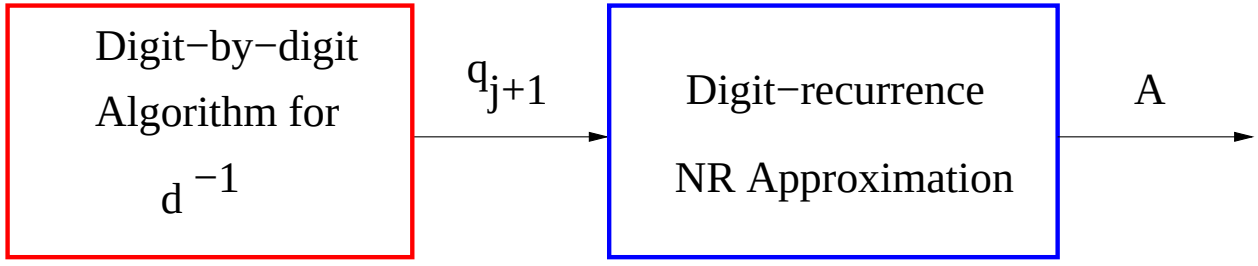
$$B = \frac{k}{2}\left(3 - dk^2\right)$$

error on k: $\delta = 1 - k\sqrt{d}$
error on B: $\epsilon = 1 - B\sqrt{d} = \frac{\delta^2}{2}\left(3 - \delta\right) \rightarrow O(\delta^2)$

Outline

- Motivation and Background
- Reciprocal Algorithm and Implementation
- Square-Root Reciprocal Algorithm and Implementation
- Evaluation and Comparisons
- Conclusions and Future Directions

Reciprocal Algorithm



$$A = Q[g](2 - dQ[g])$$

At iteration j: partial quotient and its approximation

$$Q[j] = Q[0] + \sum_{i=1}^{j} q_i r^{-i}$$

$$A[j] = Q[j](2 - d\,Q[j])$$

with error δ

with error δ^2

Reciprocal Algorithm (cont.)

- $\mathcal{W}$ recurrence computes first g quotient digits q_j

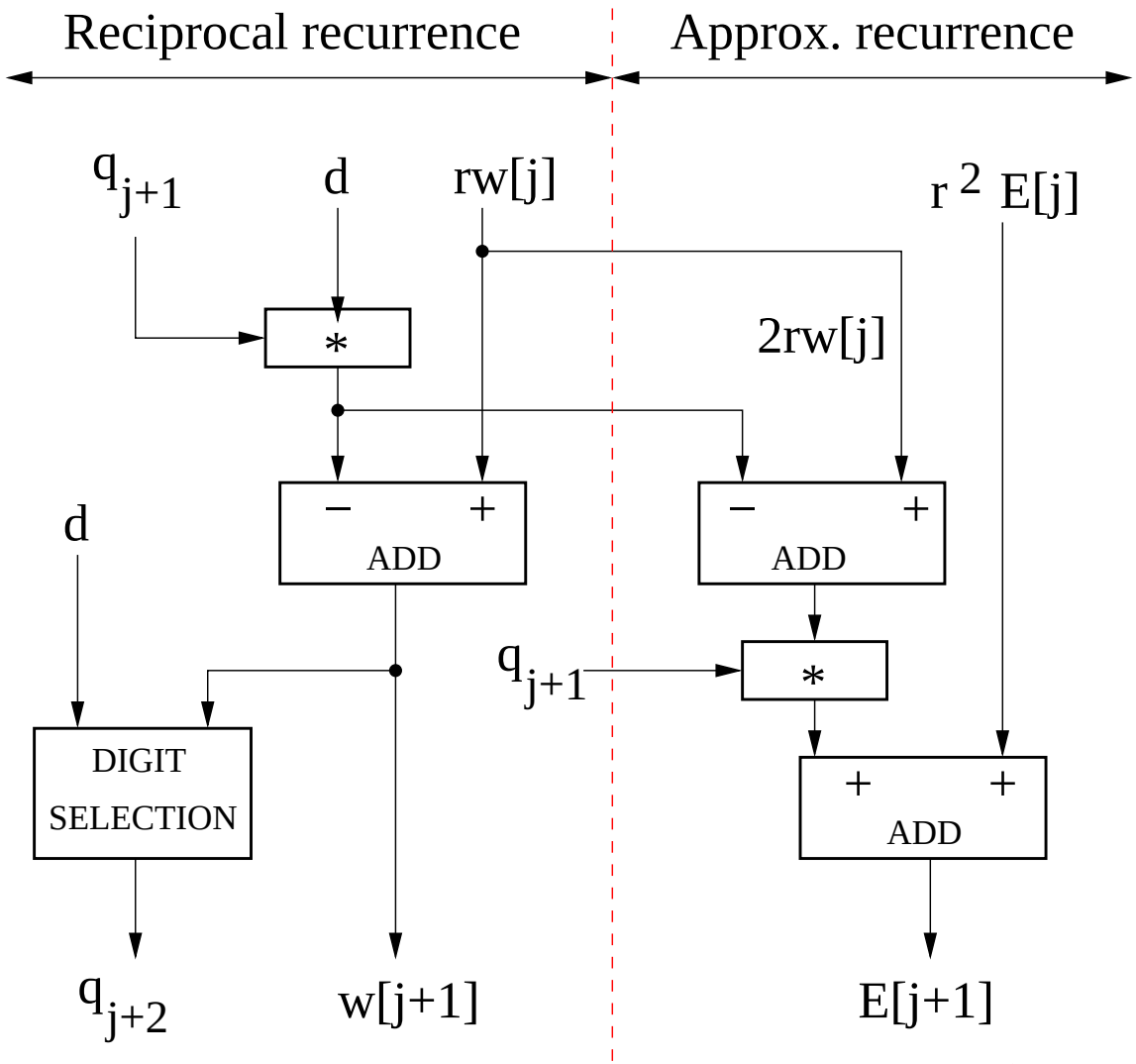
$$\begin{cases} w[j+1] &= rw[j] - q_{j+1}d \\ q_{j+2} &= SEL(rw[j+1], d) \end{cases}$$

- Approximation recurrence (overlapped)
  Introducing $E[j] = r^{2j}A[j]$

$$E[j+1] = r^2 E[j] + q_{j+1}(2rw[j] - q_{j+1}d)$$

$w[j]$ and $-q_{j+1}d$ are computed in $\mathcal{W}$ recurrence

Reciprocal Algorithm (cont.)



$$w[j+1] = rw[j] - q_{j+1}d$$

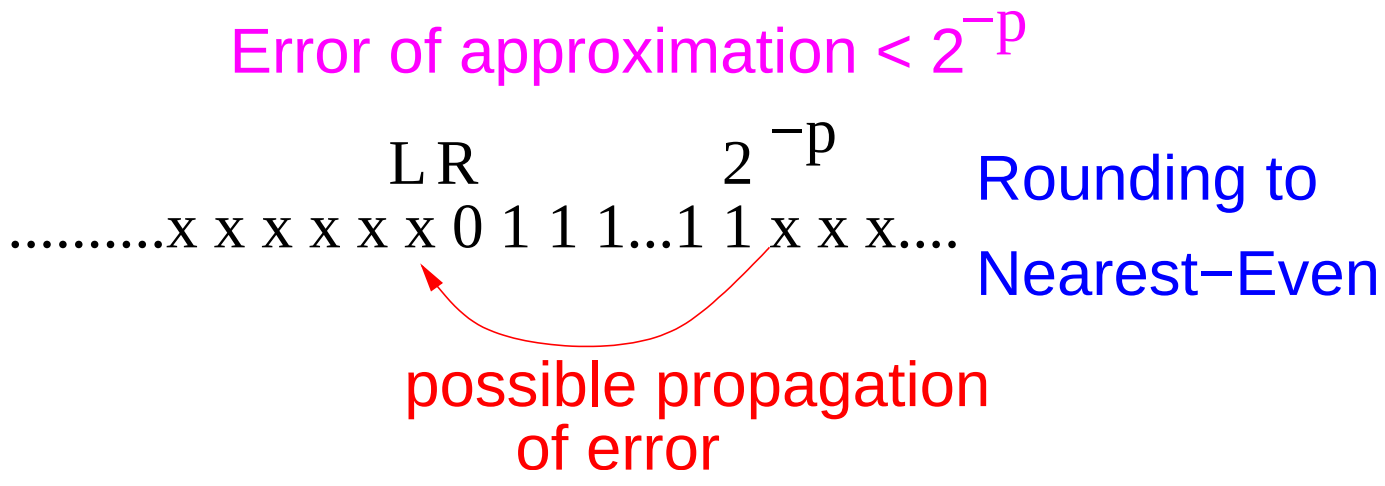
$$q_{j+2} = SEL(rw[j+1], d)$$

$$E[j+1] = r^2E[j] + q_{j+1}(2rw[j] - q_{j+1}d)$$

Exact Rounding (if needed)

Rounding consists of the following steps:

1. Determine if result obtained can be directly rounded.
   Only a few critical cases, easy to detect:



2. If not continue with the digit rec. until the rounding bit is produced (using standard recurrence)
   The latency increases for these cases
   These cases have very low probability

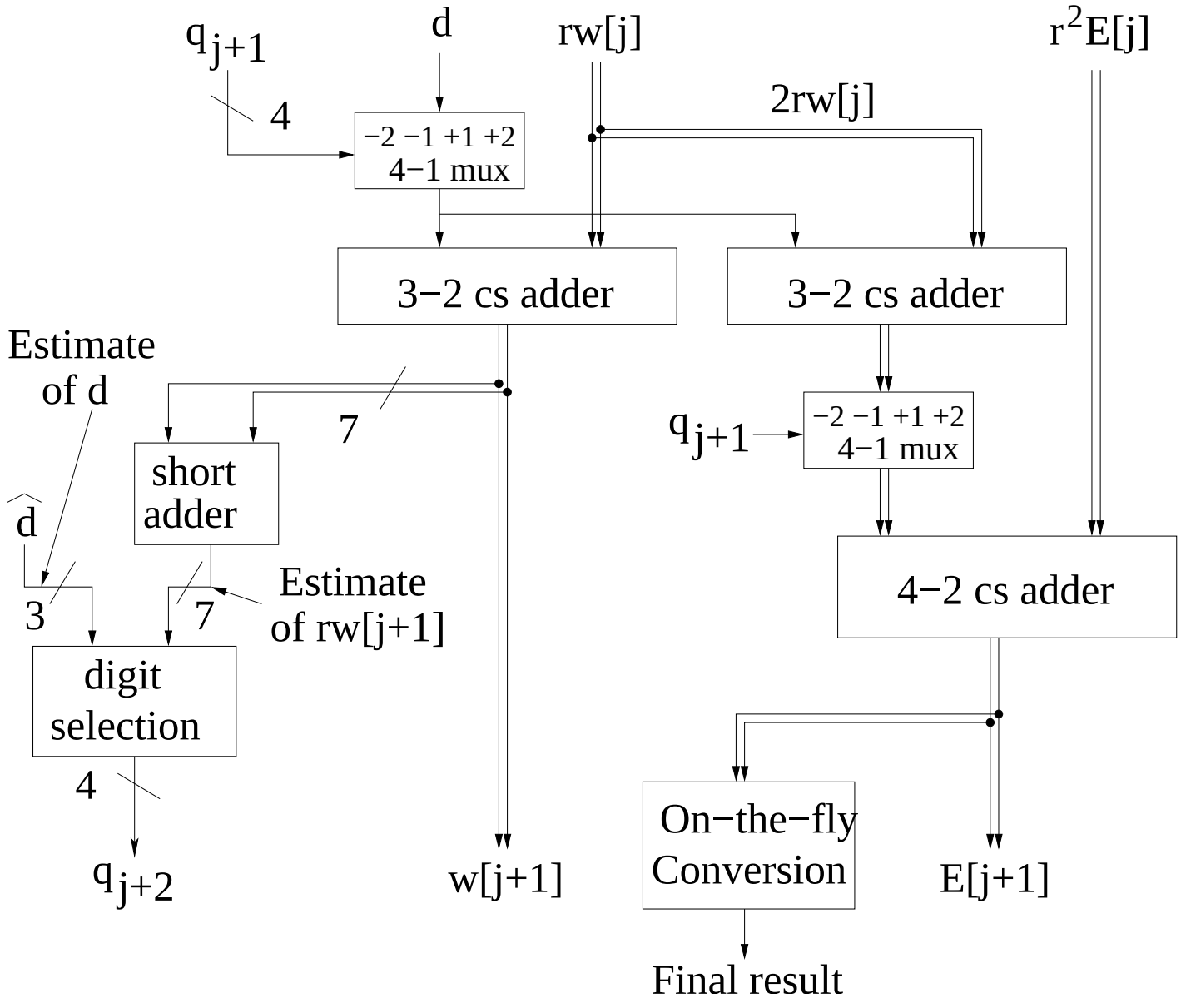
Reciprocal: Radix-4 Implementation

Recurrences and SEL

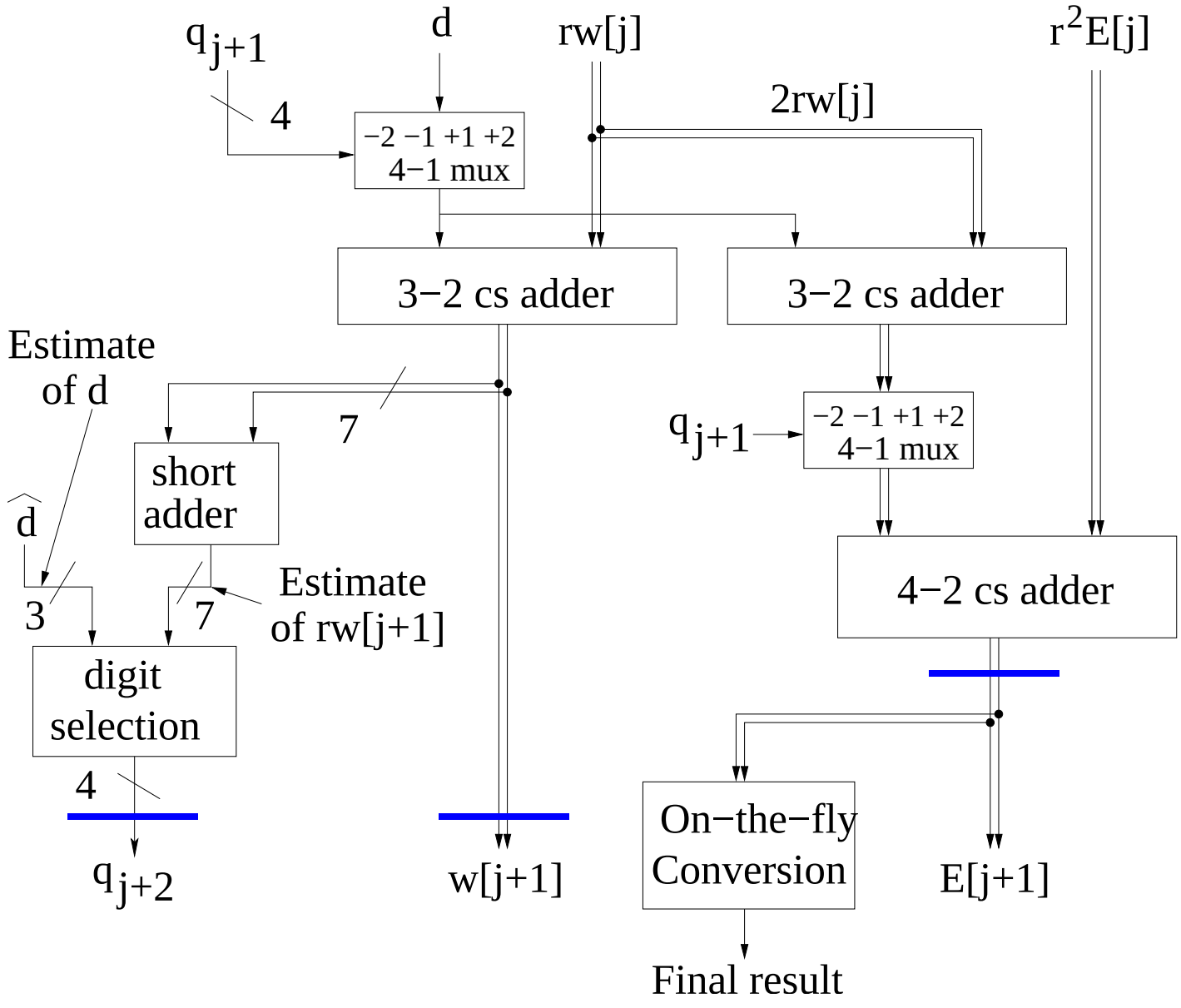
$$\begin{aligned} w[j+1] &= 4w[j] - q_{j+1}d \\ q_{j+2} &= SEL(4w[j+1], d) \\ \hline E[j+1] &= 4^2E[j] + q_{j+1}(2 \cdot 4w[j] - q_{j+1}d) \end{aligned}$$

- Iterations (recurrence) $g = 13$ (double precision)
- Quotient digit $q_j = \{-2, -1, 0, 1, 2\}$ (1-out-4 encoded)
- Selection function based on 7 bits $4w[j]$ and 3 bits d
- Carry-save representation of w and E
- Conversion done on-the-fly (1 extra cycle)

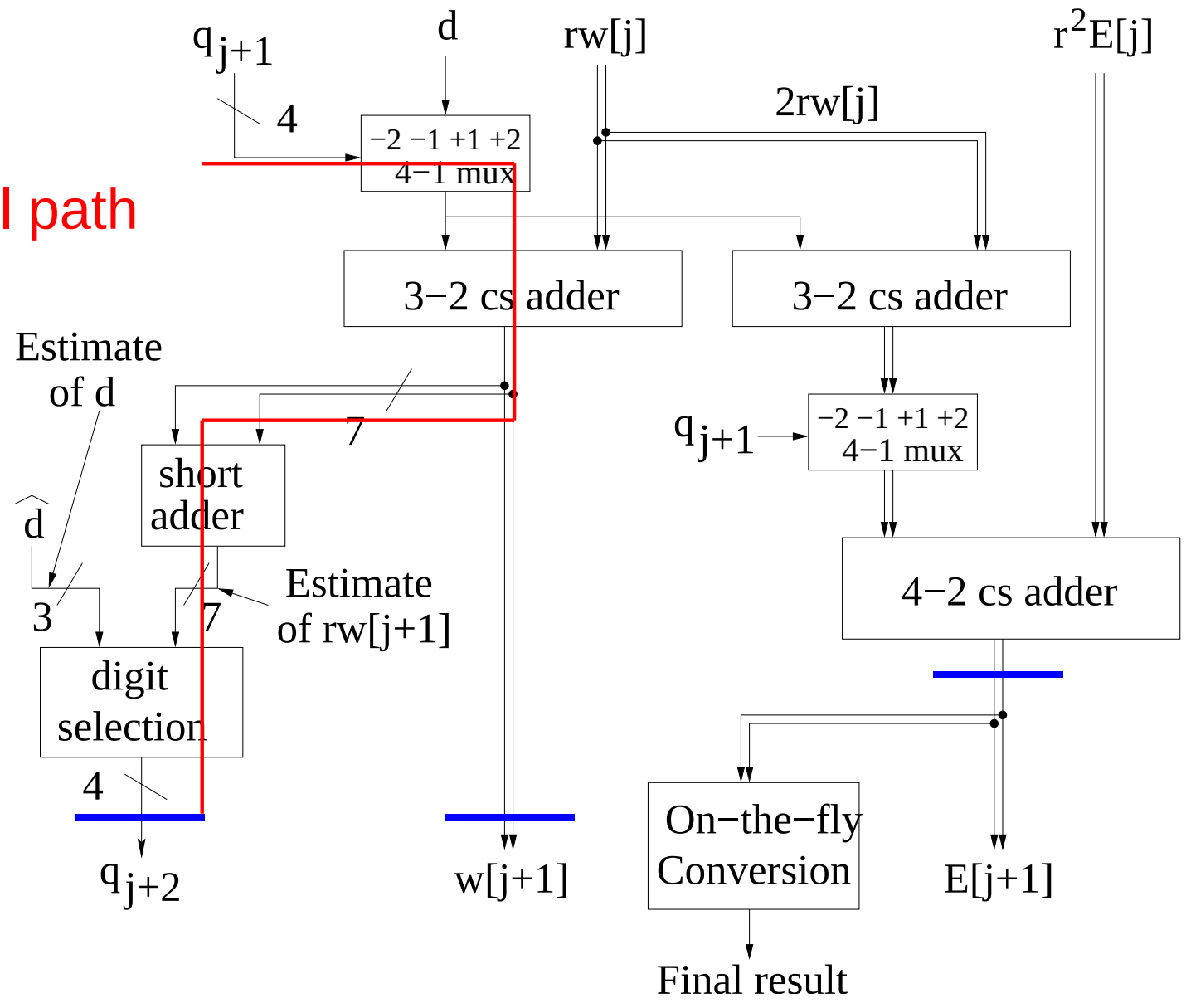
Reciprocal: Radix-4 Implementation



Reciprocal: Radix-4 Implementation



Reciprocal: Radix-4 Implementation

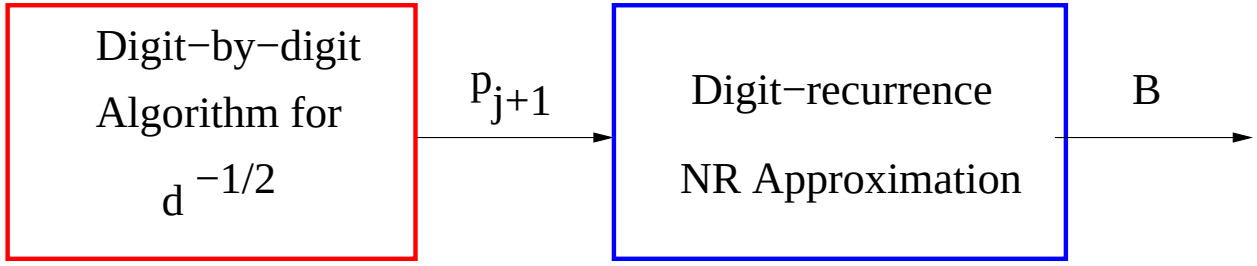


Outline

- Motivation and Background
- Reciprocal Algorithm and Implementation
- Square-Root Reciprocal Algorithm and Implementation
- Evaluation and Comparisons
- Conclusions and Future Directions

Square-Root Recipr. Algorithm

Same idea as for reciprocal but a bit more complex



$$B = P[g]\left(\frac{3}{2} - \frac{1}{2}dP[g]^2\right)$$

At iteration j: partial result and its approximation

$$P[j] = P[0] + \sum_{i=1}^{j} p_i r^{-i}$$

$$B[j] = P[j]\left(\frac{3}{2} - \frac{1}{2}dP[j]^2\right)$$

with error δ with error δ^2

To simplify: $D[j] = dP[j]$ and $C[j] = (1/2)r^{-(j+1)}d$

Square-Root Recipr. Algorithm (cont.)

- $\mathcal{W}$ recurrence computes first g result digits p_j

$$\begin{cases} w[j+1] &= rw[j] - p_{j+1}D[j] - p_{j+1}^2 C[j] \\ D[j+1] &= D[j] + 2p_{j+1}C[j] \\ C[j+1] &= r^{-1}C[j] \\ p_{j+2} &= SEL(rw[j+1], D[j+1]) \end{cases}$$

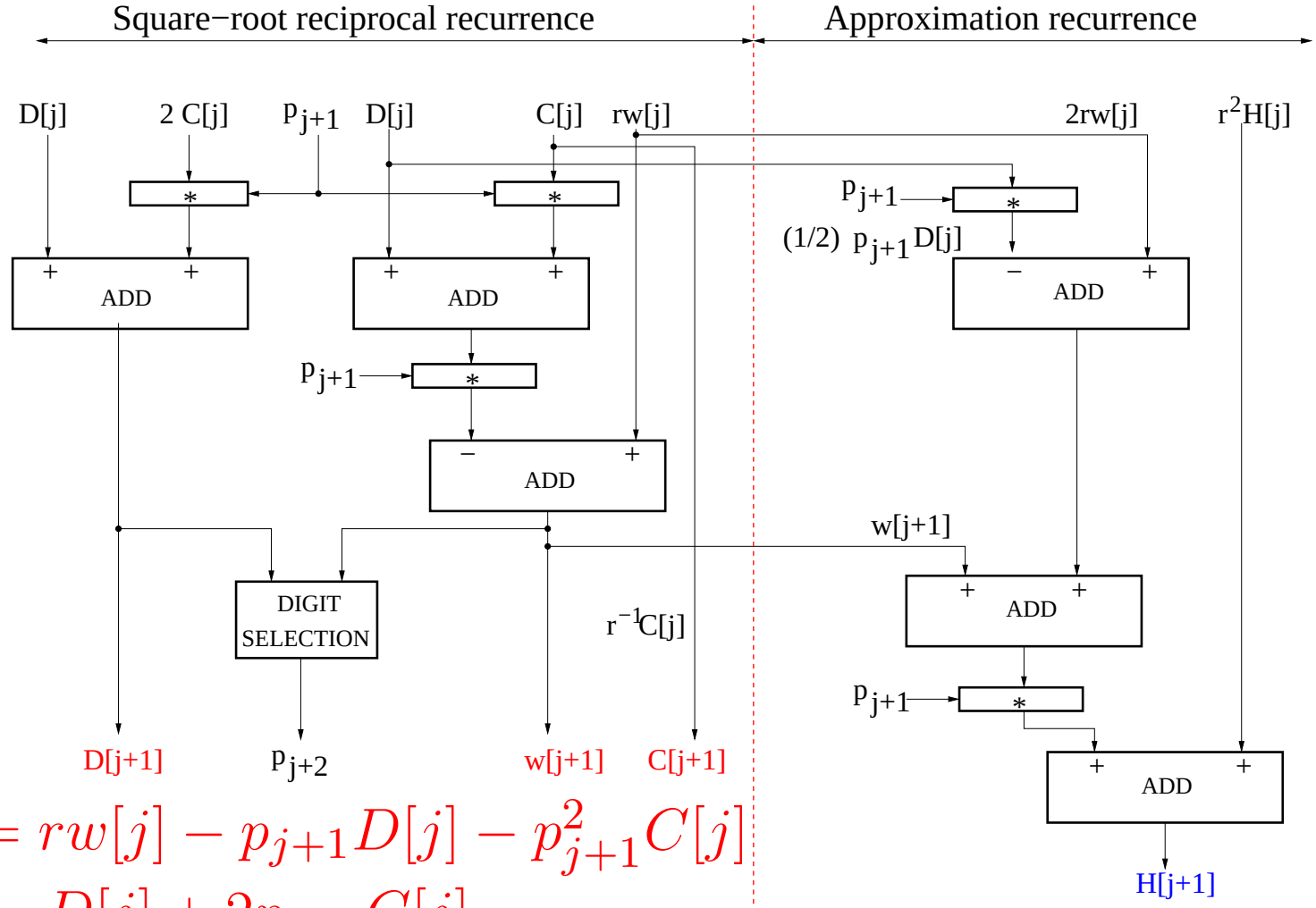
- Approximation recurrence (overlapped)
  Introducing $H[j] = r^{2j}B[j]$

  we obtain

$$H[j+1] = r^2 H[j] + p_{j+1}(2rw[j] + w[j+1] - \frac{1}{2}p_{j+1}D[j])$$

  $w[j]$, $w[j+1]$ and $D[j]$ are computed in $\mathcal{W}$ recurrence

Square-Root Recipr. Algorithm (cont.)



$$w[j+1] = rw[j] - p_{j+1}D[j] - p_{j+1}^2 C[j]$$

$$D[j+1] = D[j] + 2p_{j+1}C[j]$$

$$C[j+1] = r^{-1}C[j]$$

$$H[j+1] = r^2 H[j] + p_{j+1}(2rw[j] + w[j+1] - \tfrac{1}{2}p_{j+1}D[j])$$

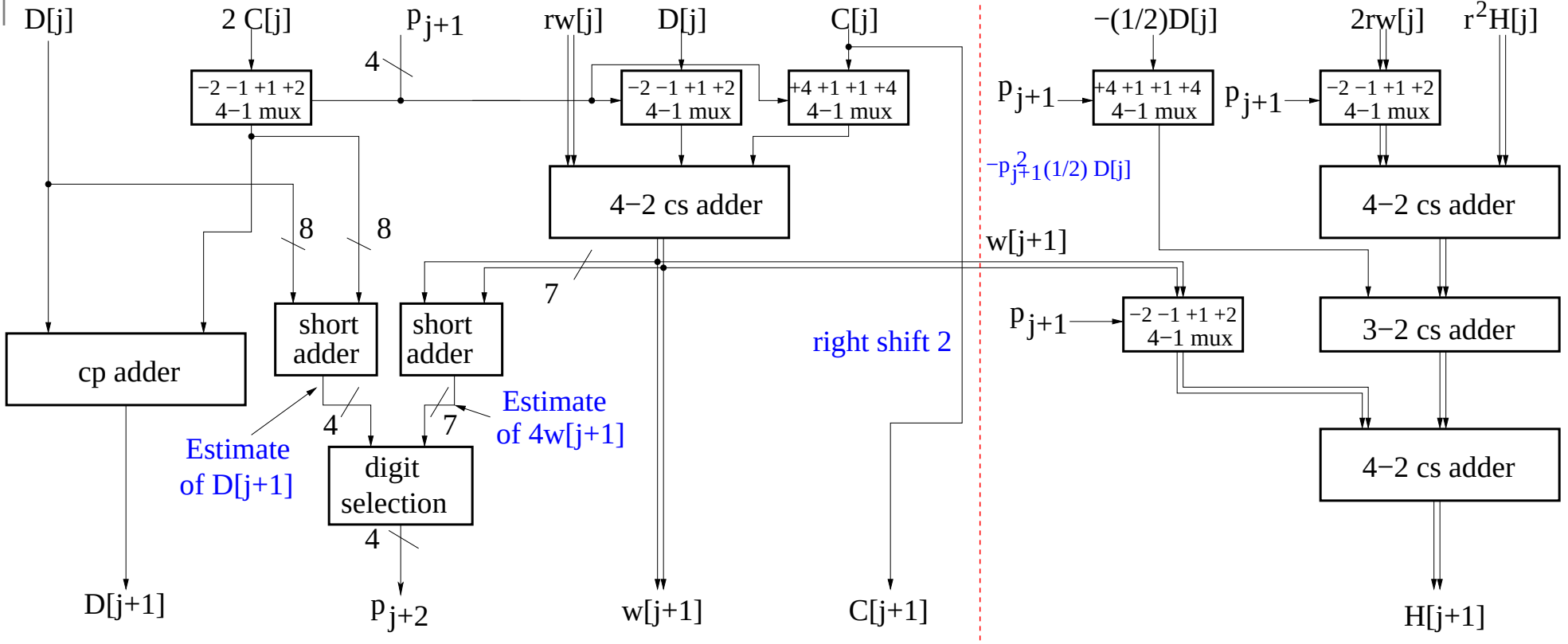
Radix-4 $\frac{1}{\sqrt{d}}$ Implementation

Recurrences and SEL:

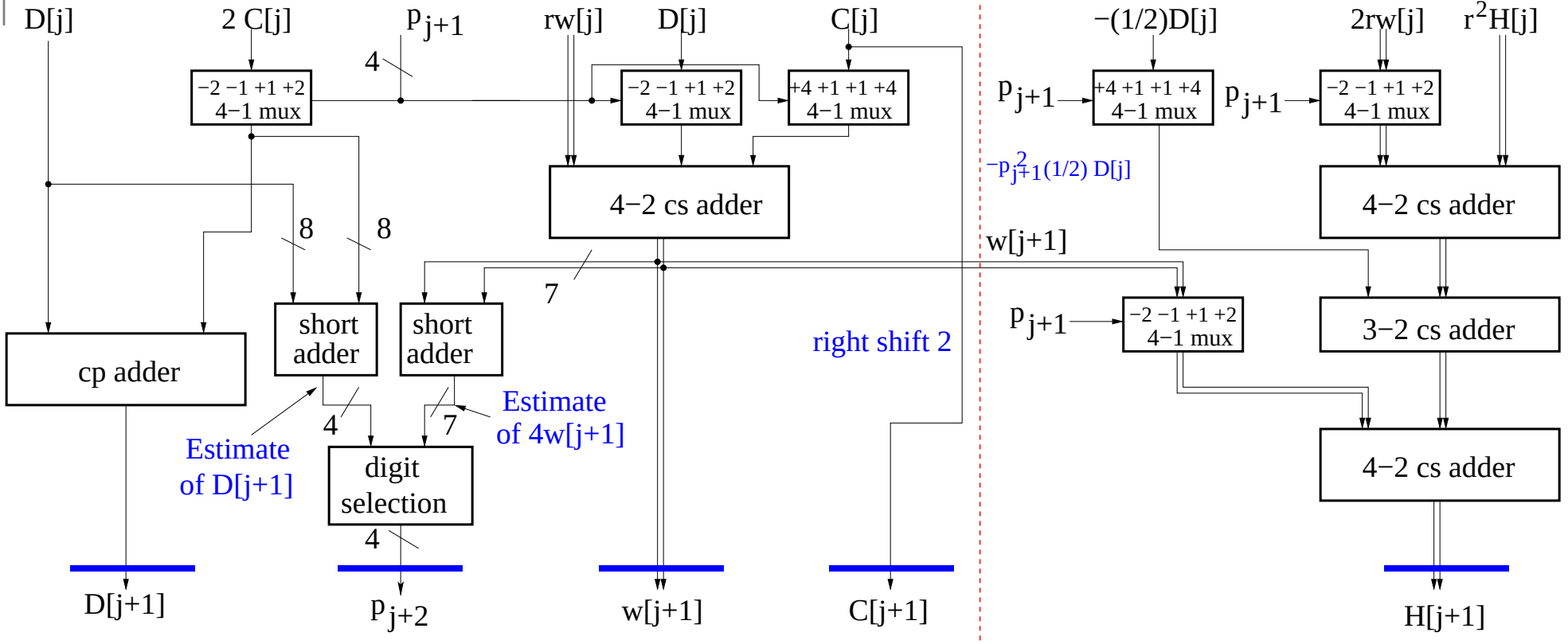
$$
\begin{aligned}
w[j+1] &= 4w[j] - p_{j+1}D[j] - p_{j+1}^2 C[j] \\
D[j+1] &= D[j] + 2p_{j+1}C[j] \\
C[j+1] &= 4^{-1}C[j] \\
p_{j+2} &= SEL(rw[j+1], D[j+1]) \\
\hline
H[j+1] &= 4^2 H[j] + p_{j+1}(2 \cdot 4w[j] + w[j+1] - \tfrac{1}{2}p_{j+1}D[j])
\end{aligned}
$$

- Iterations (recurrence) $g = 13$ (double precision)
- Quotient digit $p_j = \{-2, -1, 0, 1, 2\}$ (1-out-4 encoded)
- Selection function based on 7 bits $4w[j]$ and 4 bits $\hat{d}$
- Carry-save representation of w and H
- Conversion done on-the-fly (1 extra cycle)

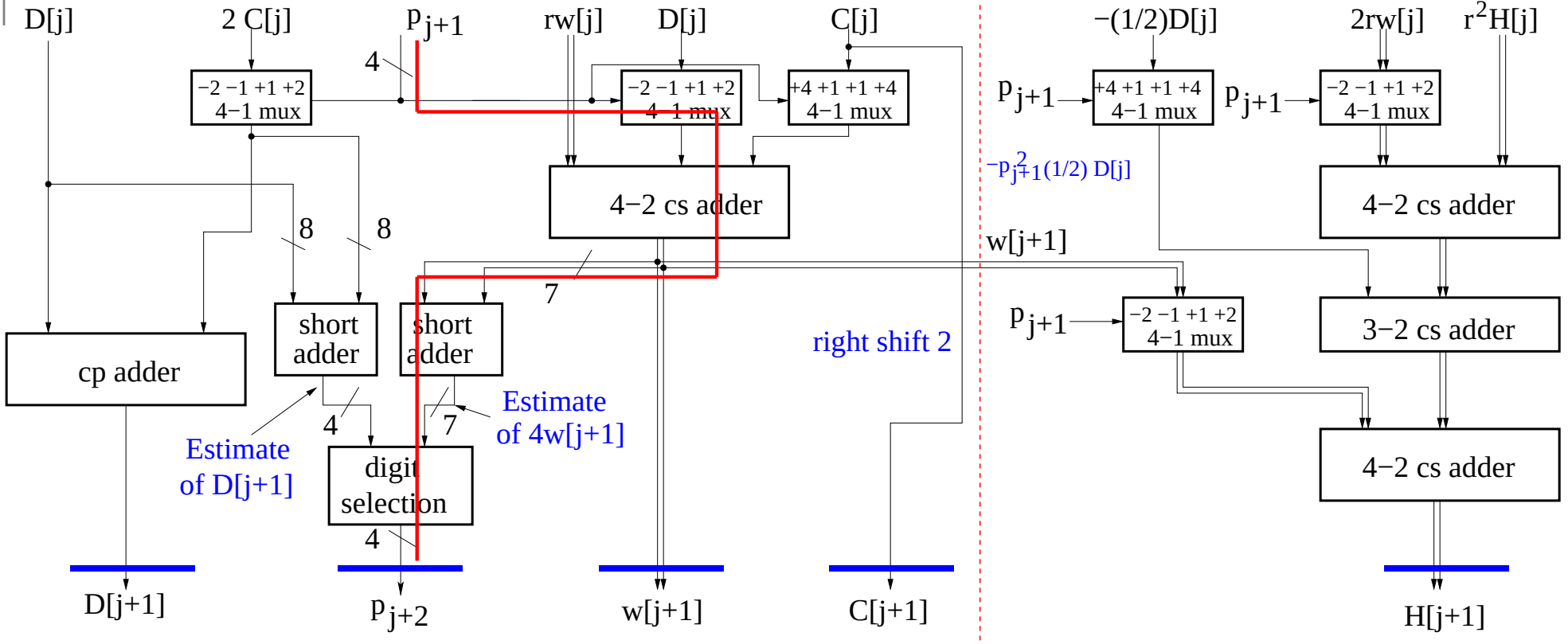
Radix-4 $\frac{1}{\sqrt{d}}$ Implementation



Radix-4 $\frac{1}{\sqrt{d}}$ Implementation



Radix-4 $\frac{1}{\sqrt{d}}$ Implementation



critical path

Outline

- Motivation and Background
- Reciprocal Algorithm and Implementation
- Square-Root Reciprocal Algorithm and Implementation
- Evaluation and Comparisons
- Conclusions and Future Directions

Implementation Results

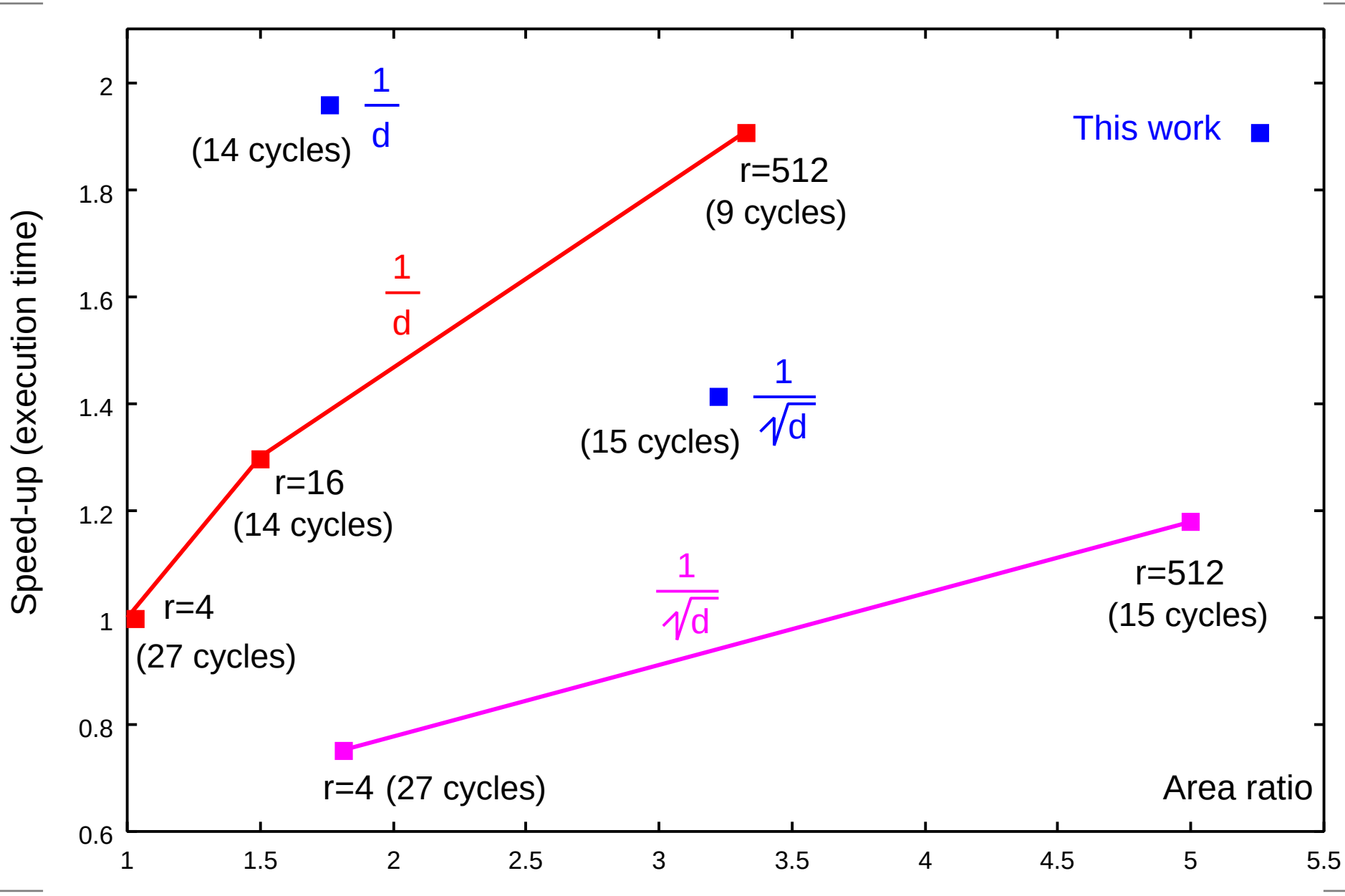
$1/d$ and $1/\sqrt{d}$ units implemented in $0.18\ \mu m$ Standard Cells

Unit	Critical path						
$1/d$	t_{buf}	t_{mux}	t_{xor3}	t_{sel}	t_{reg}		
	0.11	$+\ 0.20$	$+\ 0.16$	$+\ 0.61$	$+\ 0.28$	$=$	$1.36\ ns$
$1/\sqrt{d}$	t_{buf}	t_{mux}	t_{4-2}	t_{sel}	t_{reg}		
	0.05	$+\ 0.20$	$+\ 0.27$	$+\ 0.97$	$+\ 0.28$	$=$	$1.77\ ns$

Cycle time of $1/d$ same as r-4 digit rec. ($\sim$ 20 $t_{NOT-FO4}$)

Unit	Area			
	D-D rec.	Approx.	Conv.	Total
$1/d$	2700	4550	3500	11650
$1/\sqrt{d}$	8200	7700	3500	19500

Comparison other Digit-recurrence Impl.



Outline

- Motivation and Background
- Reciprocal Algorithm and Implementation
- Square-Root Reciprocal Algorithm and Implementation
- Evaluation and Comparisons
- Conclusions and Future Directions

Conclusions

The combination of $1/d$ ($1/\sqrt{d}$) digit-by-digit algorithm and approximation allows

- Quadratic convergence and halving of execution time (same cycle time)
- Correct rounding with small average overhead
- Speed-up 1.5 with respect to r-16 (area increase 20%)

Future Work

- Explore implementations for higher radices
- Extension to obtain also division and square root
- Check the use of faster r-4 digit recurrence (selection by comparison)